MAGAZINE

BSD

FOR NOVICE AND ADVANCED USERS

Basic UNIX Commands

HOW TO TEST THE SECURITY OF THE OPERATING SYSTEM

GETTING TO GRIPS WITH THE GIMP – PART 7

FLAWS IN UNIX-LIKE ROOTKITS AND ANTI-ROOTKIT TOOLS

INTRODUCTION TO THE JDB DEBUGGER

VOL.8 NO.09
ISSUE 09/2014(62)
1898-9144

FREENAS MINI
STORAGE APPLIANCE

IT *SAVES* YOUR LIFE.

HOW IMPORTANT IS YOUR DATA?

Years of family photos. Your entire music and movie collection. Office documents you've put hours of work into. Backups for every computer you own. We ask again, *how important is your data?*

NOW IMAGINE LOSING IT ALL

Losing one bit - that's all it takes. One single bit, and your file is gone.

The worst part? **You won't know until you absolutely need that file again.**

Example of one-bit corruption

THE SOLUTION

The FreeNAS Mini has emerged as the clear choice to save your digital life. **No other NAS in its class offers ECC (error correcting code) memory and ZFS bitrot protection to ensure data always reaches disk without corruption and *never degrades over time*.**

No other NAS combines the inherent data integrity and security of the ZFS filesystem with fast on-disk encryption. No other NAS provides comparable power and flexibility. The FreeNAS Mini is, hands-down, the best home and small office storage appliance you can buy on the market. **When it comes to saving your important data, there simply is no other solution.**

The Mini boasts these state-of-the-art features:

- 8-core 2.4GHz Intel® Atom™ processor
- Up to 16TB of storage capacity
- 16GB of ECC memory (with the option to upgrade to 32GB)
- 2 x 1 Gigabit network controllers
- Remote management port (IPMI)
- Tool-less design; hot swappable drive trays
- FreeNAS installed and configured

http://www.iXsystems.com/mini

FREENAS CERTIFIED STORAGE

With over six million downloads, FreeNAS is undisputedly *the* most popular storage operating system in the world.

Sure, you could build your own FreeNAS system: research every hardware option, order all the parts, wait for everything to ship and arrive, vent at customer service because it *hasn't*, and finally build it yourself while hoping everything fits - only to install the software and discover that the system you spent *days* agonizing over **isn't even compatible**. Or...

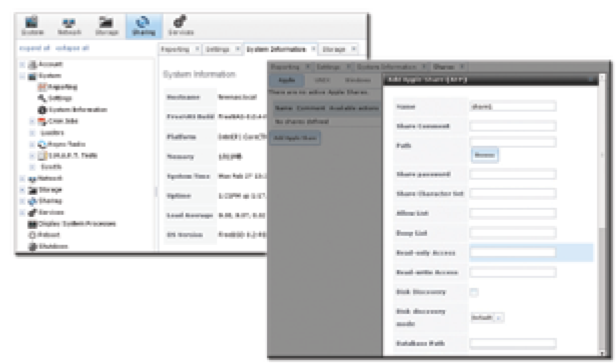
MAKE IT EASY ON YOURSELF

As the sponsors and lead developers of the FreeNAS project, iXsystems has combined over 20 years of hardware experience with our FreeNAS expertise to bring you FreeNAS Certified Storage. **We make it easy to enjoy all the benefits of FreeNAS without the headache of building, setting up, configuring, and supporting it yourself.** As one of the leaders in the storage industry, you know that you're getting the best combination of hardware designed for optimal performance with FreeNAS.

Every FreeNAS server we ship is...

» Custom built and optimized for your use case
» Installed, configured, tested, and guaranteed to work out of the box
» Supported by the Silicon Valley team that designed and built it
» Backed by a 3 years parts and labor limited warranty

As one of the leaders in the storage industry, you know that you're getting the best combination of hardware designed for optimal performance with FreeNAS. **Contact us today for a FREE Risk Elimination Consultation with one of our FreeNAS experts.** Remember, every purchase directly supports the FreeNAS project so we can continue adding features and improvements to the software for years to come. **And really - why would you buy a FreeNAS server from *anyone* else?**



FreeNAS 1U

- Intel® Xeon® Processor E3-1200v2 Family
- Up to 16TB of storage capacity
- 16GB ECC memory (upgradable to 32GB)
- 2 x 10/100/1000 Gigabit Ethernet controllers
- Redundant power supply

FreeNAS 2U

- 2x Intel® Xeon® Processors E5-2600v2 Family
- Up to 48TB of storage capacity
- 32GB ECC memory (upgradable to 128GB)
- 4 x 1GbE Network interface (Onboard) - (Upgradable to 2 x 10 Gigabit Interface)
- Redundant Power Supply

http://www.iXsystems.com/storage/freenas-certified-storage/

Dear Readers,

BSD magazine has released many useful and practical articles to help you improve your skills since our first published issue. I hope that we are still able to teach you and provide you with much-needed knowledge, so I'd like to encourage you to tell us what your IT needs are.

Let's take a look at what you will find in the September issue of BSD magazine. Our expert will introduce you to the products, technologies and protocols which will be used in setting up an office server. You will discover the top 100 commands in Unix/Linux that an attacker (and most crucially, a security tester) can use. Finally, you may be interested in the JDB tool which is a command line debug tool that will help you find bugs in your code (it is provided by openjdk).

I would like to express my gratitude to our experts who contributed to this publication and invite others to cooperate with our magazine. The next issue of BSD will be published in 4 weeks.

If you are interested in learning more about future content, our BSD workshops specifically, or if you would like to get in touch with our team, I would like to introduce Gurkan Mercan to you. You can reach him at *gurkan.mercan@bsdmag.org*.

Hope you enjoy the issue.

Ewa Dudzic
ewa.d@bsdmag.org

MAGAZINE BSD

Editor in Chief:
Ewa Dudzic
ewa.dudzic@software.com.pl

Contributing:
Michael Shirk, Andrey Vedikhin, Petr Topiarz, Charles Rapenne, Anton Borisov, Jeroen van Nieuwenhuizen, José B. Alós, Luke Marsden, Salih Khan, Arkadiusz Majewski, BEng, Toki Winter, Wesley Mouedine Assaby, Rob Somerville

Top Betatesters & Proofreaders:
Annie Zhang, Denise Ebery, Eric Geissinger, Luca Ferrari, Imad Soltani, Olaoluwa Omokanwaye, Radjis Mahangoe, Mani Kanth, Ben Milman, Mark VonFange

Special Thanks:
Annie Zhang
Denise Ebery

Art Director:
Ireneusz Pogroszewski

DTP:
Ireneusz Pogroszewski
ireneusz.pogroszewski@software.com.pl

Senior Consultant/Publisher:
Paweł Marciniak
pawel@software.com.pl

CEO:
Ewa Dudzic
ewa.dudzic@software.com.pl

Publisher:
Hakin9 Media SK
02-676 Warsaw, Poland
Postepu 17D
Poland
worldwide publishing
editors@bsdmag.org
www.bsdmag.org

Hakin9 Media SK is looking for partners from all over the world. If you are interested in cooperation with us, please contact us via e-mail: editors@bsdmag.org.

FreeNAS

in an Enterprise Environment

By the time you're reading this, FreeNAS has been downloaded more than 5.5 million times. For home users, it's become an indispensable part of their daily lives, akin to the DVR. Meanwhile, all over the world, thousands of businesses universities, and government departments use FreeNAS to build effective storage solutions in myriad applications.

What you will learn...

- How TrueNAS builds off the strong points of the FreeBSD and FreeNAS operating systems
- How TrueNAS meets modern storage challenges for enterp

The FreeNAS operating systems is fre
the public and offers thorough doc
active community, and a feature-ri
the storage environment. Based on Free
can share over a host of protocols (SMB
FTP, iSCSI, etc) and features an intuitiv
the ZFS file system, a plug-in system
much more.

Despite the massive popularity
aren't aware of its big brother duti
data in some of the most demand
environments: the proven, enterp
professionally-supported line of

But what makes TrueNAS diffe
Well, I'm glad you asked...

Commercial Grade Supp

When a mission critical stor
organization's whole operat
halt. Whole community-bas
free), it can't always get an
and running in a timely m
responsiveness and expe
dedicated support team
provide that safety.

Created by the sam
developed FreeNAS.

FreeBSD Technologies and Protocols Involved in Setting up an Office Server

The goal of this article is to introduce you to the products, technologies and protocols which will be used in setting up an office server. The article will describe them in detail in order to clarify their interaction with the server and with the Internet at large.

The FreeBSD operating system is the layer of software which exists between applications and the hardware. Each low-level operation issued by the application (such as reading or writing a file) will be transformed by the operating system's kernel into a series of instructions to the hardware (such as finding the correct place on the hard drive to read or write and programming the correct hardware controllers to perform the operation).

There are many features of a modern computer environment which users take for granted, but which are only possible because of advanced algorithms implemented in the operating system's kernel, such as: multitasking and switching between applications, security isolation between users and between applications, network communication between distant systems, memory management and swapping.

As traditional for modern BSD operating systems, FreeBSD is divided into at least three distinct parts: the kernel, the userland utilities, and the third party application collection.

The kernel is the really low-level interface to the hardware. The only way an application can run is by being started by the kernel and by using kernel services to do all of the non-trivial tasks, like communicating with the user (either graphically or textually), reading or writing files, or using the network.

New applications are started when existing applications request that the kernel load and execute specific files, and the kernel cleans up the resources used by the application after it is done executing.

UserLand

The term "userland" refers to all the applications running on top of a kernel. The userland utilities of the operating system are present for convenience to the system administrator and its users. They are in fact a common, more-or-less standardised way of configuring and managing a system. For example: though the kernel has a concept of different users which may or may not be forbidden to read each other's files, there needs to be a set of applications which manage a list of these users.

The kernel knows how to read and write data to and from files, but it needs to be externally configured with which devices contain which file systems and where to mount these. As another example: the kernel knows how to send, receive and route network packets, but there

needs to be a way to tell the kernel that certain network interfaces (or "network cards") have certain IP addresses, what the default gateway is, and which networks, if any, are connected to this one.

All of these tasks, and plenty more, are performed by the operating systems' userland utilities. The FreeBSD base system contains enough utilities to make it a complete application development environment for the operating system itself, including a compiler and a text editor.

There is a sharp distinction between program code that is a part of the kernel and that which is a part of a regular application (or "userland"). The userland code cannot access the hardware directly but must rely on the kernel to provide its services, but because of this the applications can focus on performing some high-level tasks useful to the users, and can grow quite large.

The kernel code exists mostly to provide services to userland applications so it must be fast and compact, but at the same time since it is "close" to the hardware it is bogged down by the fact that it needs to perform many very low-level operations by itself.

In contrast to some operating systems (most notably Linux), the BSD userland utilities are maintained by the same group of people and in the same project package as the kernel.

This makes the utilities leaner and allows deeper integration between the features provided by the kernel and the utilities. In FreeBSD (and most other BSDs), the collection of the userland utilities shipped with the operating system is sometimes called the "base system" or the "world".

The collection of third party applications is what makes an operating system generally usable. While basic admin-

istration tools such as those to manage users, files, file systems, and the network are enough to have a running, working system, it is the third party applications which usually provide useful user-visible features to this system.

Those can be web servers, databases, e-mail servers, desktop environments, games and multimedia applications.

The manual pages

FreeBSD comes with an extensive set of system manual pages. These cover all aspects of the system: the kernel and its facilities, the userland utilities and even the programming APIs and libraries from which the system is built.

These manual pages are grouped into sections identified by their number:

- General commands – such as ls, mkdir, top, ps, ssh, etc.
- System calls (the kernel API) – such as `open()`, `close()`, `read()`, `write()`, `socket()`, etc.
- C library functions – such as `strcpy()`, `err()`, `strtod()`, `malloc()`, `pthread_create()`, etc.
- Device drivers and kernel interfaces – such as ahci, ehci, vga, ipfw, tun, ipsec, etc.
- Configuration files and file formats – such as rc.conf, jail.conf, crontab, fstab, etc.
- “Games” (non-essential software) – such as fortune, morse, ppt, etc.
- “Miscellaneous information” (documentation) – such as hier, ports, tuning, etc.
- System management utilities – such as adduser, fsck, newfs, syslogd, gmirror, etc.
- Kernel internal API (“KPI”) – such as mbuf, signal, wakeup, rtalloc, etc.

There is no logical relation between the number and the section content – the assignment was historical and evolutionary, not intelligently designed.

A manual page can be viewed in the terminal by issuing a command such as:

```
> man pkg
```

The manual viewer (“man”) will find and display the named manual page from the lowest-numbered section which contains the page with the given name. In case there are manual pages with the same name in different sections (for example: ipfw is both a kernel interface and a userland utility to manage this kernel interface), you can force the choice of the section by adding it before the page name:

```
> man 8 ipfw
```

In official FreeBSD documents, “man page” references are usually written with the page name followed by the section number in parenthesis, such as “ipfw(8)”.

Ports and packages

The traditional way third party applications are introduced into a BSD operating system is via a variation of a concept called the “ports tree.”

This is a structure of directories (“folders”) containing scripts and other utilities which compile and install specific pieces of software.

This was an efficient and agile method of making software available to the users because only this directory tree needed to be distributed – the actual application source code was automatically fetched just before it was compiled. It enabled quick adoption of new software and new versions of existing applications.

However, the principle of having to compile source code for every application which needs to be installed or updated is not very scalable at the present time where the applications required on single systems are more numerous and time is at a premium. Compiling large desktop software such as LibreOffice can take hours, or even days on low-end hardware. On the other hand, only by compiling software from scratch can the users have direct control over various optional features of such software.

Since FreeBSD 10, the preferred method of installing and maintaining software is with the “pkg” utility, which deals with binary (pre-compiled) software packages.

The ports tree is still present and it is the source from which the binary packages are built, but installing software from both the ports tree and the binary packages can result in problems.

The package manager

The “pkg” package manager shares much of its design with similar software on Linux and other operating systems. It maintains databases of installed and available application packages, and enables the user to search, install, remove or query packages.

On the command line, pkg is invoked by passing a command and optional command arguments to the main program, for example:

```
> pkg search vim-lite
# pkg install vim-lite
> pkg info -f vim-lite
# pkg remove vim-lite
```

A freshly installed FreeBSD system only comes with a small bootstrap program instead of the full package manager.

This program will install the package manager on first invocation of the pkg command.

Installing software

The first step for installing software usually involves finding it. The "search" argument to the "pkg" command followed by a string of letters will result in a print-out of all the packages available in the package repositories whose names contain the specified string. For example:

```
> pkg search dovecot
currently lists the following packages:
dovecot-1.2.17_4
dovecot-antispam-1.3_2,1
dovecot-managesieve-0.11.13_1
dovecot-pigeonhole-0.4.3_1
dovecot-sieve-1.2+0.1.19_1
dovecot2-2.2.13_3
dovecot2-antispam-plugin-20130429_7
```

There are often multiple versions of the same applications offered in the repository. It is up to you to select the required version. After choosing the correct software (for example, in this case, let's say it's dovecot2), the software can be installed by issuing a command such as:

```
> pkg install dovecot2
```

More than one name can be added after the "install" argument, and the package manager will find and install all the packages and their dependencies in the same operation.

De-installing (removing) software

Removing packages also usually starts with finding a package to remove. In this case we will utilize the "info" argument to the "pkg" command with the "-g" switch to find all packages matching a pattern in the "shell glob" format:

```
> pkg info -g 'php5*'
```

Note

Don't forget to add the single quotes (the apostrophes) around the shell glob expression, to prevent your current shell from expanding the glob expression itself.

The above command will list all packages beginning with 'php5'. An example of its output is:

```
php5-5.4.23
php5-bz2-5.4.21
php5-calendar-5.4.21
php5-ctype-5.4.21
php5-curl-5.4.21_1
php5-dom-5.4.21
php5-exif-5.4.21
php5-filter-5.4.23
php5-ftp-5.4.21
php5-gd-5.4.21
php5-hash-5.4.21
php5-iconv-5.4.21
php5-json-5.4.21
```

The "delete" argument to "pkg" will remove the named package:

```
> pkg delete php5-ftp
```

The "pkg" command implements a large number of arguments dealing with specific situations, and its manual pages are fairly complete, so it can be informative to read through them.

Web servers and application servers

One of the first steps after installing FreeBSD is to install Apache and PHP. Apache is a web server, one of many existing today. It is a piece of software which implements the HTTP protocol which is used by web browsers to access web pages.

By default, software such as Apache only serves ordinary files from the file system, so called "static files". These can include complex HTML pages but also images, CSS and JavaScript files, MP3 files, WebM videos, etc.

In order to have a complex web application, an additional piece of software is needed: an application server. This is software which runs server-side code to e.g. process content before it is served to the web browser, handle user logins, or generate new, dynamic content.

One of the most popular application servers is PHP, which will also be used in this tutorial to run some specific services.

The web server and the application need to communicate in some way, and one of the standard ways to do this is via the FastCGI protocol.

This protocol enables the use of PHP as a proper application server in a separate process, which is beneficial from the aspects of performance and security. Our specific setup will use Apache version 2.4 and the mod_fcgid module to interface it with PHP.

An additional component now required in non-trivial web server setups: the SSL / TLS encryption layer ("SSL" is an obsolete name for the set of protocols now called "TLS").

This is a protocol which lays below HTTP and ensures that all communication between the web browser and the web server is encrypted and secured. An important aspect of SSL is that it requires a type of server credentials: there needs to be a way for the server to introduce itself to the client (the browser) and for the client to trust this introduction.

This is accomplished by using so-called “SSL certificates” which use public-key cryptography to implement a trust model where a third party (the “certificate authority”) vouches that it has issued a certificate for a specific server and knows about it.

The browsers carry a list of trusted certificate authorities which they use to check certificates sent by the servers as an introduction. Modern web servers support SSL / TLS out of the box, but they need to be configured with the valid SSL certificate and associated data in order to function. The combination of SSL / TLS and the HTTP protocol is called HTTPS.

File sharing in the local network

A common goal when connecting computers directly in a network is to enable their users to share files. The file sharing service of networks is so important that there are a large number of different protocols and ways to accomplish this, but the most common ones today are SMB / CIFS and NFS.

The names “SMB” and “CIFS” usually refer to the same protocol, though technically CIFS is a subset of SMB. It is a protocol developed and updated by Microsoft which gained popularity together with its Windows operating system, and is now available in most other operating systems, including OS X and Linux. Its biggest advantage is that it integrates seamlessly with Windows workstations.

The Samba project delivers a robust implementation of a SMB / CIFS server which can even act as a Domain Controller or an Active Directory controller. Its disadvantage is that its security model and the way users and groups are handled is at odds with how they are handled in Unix-like operating systems, so there are limitations and compatibility tweaks needed to maintain appropriate levels of interoperability. If the computers involved in file sharing are Unix-like, they can use the NFS protocol. It is a light-weight protocol developed specifically for Unix-like systems so the interoperability is much better.

It can be used natively with FreeBSD, OS X and Linux, but has a slight disadvantage in available security settings, since it is made for completely trusted networks where computers can be permitted to access each-other’s files based on nothing but having a certain IP address (no usernames or passwords are exchanged).

Despite this, it is very common and the de-facto standard for local file sharing on Unix-like systems.

The complexities of e-mail

E-mail is a hugely popular Internet service, but it is also one of its most complex. A part of the reason for this complexity is that it is one of the earliest services which has survived essentially intact in the modern era. The backbone of the e-mail service is the SMTP protocol, used simply to pass e-mail messages from server to server.

The protocol itself and the messages it carries contain next to no security provisions, making e-mail spam and forgery an everyday occurrence.

In addition to this, a completely different type of protocol, like IMAP, is required to retrieve the messages from the servers.

In order for e-mail to function, the SMTP server (which commonly contains the users’ mailboxes) should have a fixed IP address and an appropriate entry in the DNS system needs to be created to point to it.

Next, the SMTP server needs to be properly configured to only accept messages addressed to a certain domain, to be resistant to various attacks, and to perform at least some kind of spam checking of the messages before allowing them to be passed through to the users. A separate IMAP server needs to be configured to enable users to retrieve their messages from the server’s mailboxes.

All these protocols should be secured with SSL, similarly to the web servers.

Conclusions

Generally, we use Postfix 2.11 as the SMTP server and Dovecot 2.2 for the IMAP server. POP3 will not be configured, as it is an example of a very obsolete protocol. In addition to those, Spamassassin will be used to perform spam analysis and the RoundCube application will be used to provide web-based access to e-mail.

If you want to learn more, I invite you to my workshop on *Deploying an office / workgroup server on FreeBSD*.

IVAN VORAS

Ivan Voras is a FreeBSD developer and a long-time user, starting with FreeBSD 4.3 and throughout all the versions since. In real life he is a researcher, system administrator and a developer, as opportunity presents itself, with a wide range of experience from hardware hacking to cloud computing. He is currently employed at the University of Zagreb Faculty of Electrical Engineering and Computing and lives in Zagreb, Croatia. You can follow him on his blog in English at http://ivoras.net/blog or in Croatian at http://hrblog.ivoras.net/, as well as on Google+ at https://plus.google.com/+IvanVoras.

Introduction to the JDB Debugger

"The Java Debugger (JDB) is a simple command-line debugger for Java classes. The jdb command and its options call the JDB. The jdb command demonstrates the Java Platform Debugger Architecture (JDBA) and provides inspection and debugging of a local or remote Java Virtual Machine (JVM). See Java Platform Debugger Architecture (JDBA) at http://docs.oracle.com/javase/8/docs/technotes/guides/jpda/index.html.

JDB, like GDB, is a command line debug tool that will help us find bugs in our code (it is provided by openjdk), hopefully with little effort. The application that we'll be using to demonstrate JDB's usefulness will follow the same settings as the last one. The application we are going to use is Freecol:

"The FreeCol team aims to create an Open Source version of Colonization (released under the GPL). At first we'll try to make an exact clone of Colonization. "

Like the last time we used GDB, we need the sources of the freecol application to be compiled with extra debugging symbols (-g flag to the java compiler). Requirements:

- You must download the following package: *http://prdownloads.sourceforge.net/freecol/freecol-0.10.7-src.zip?download*.
- You must have the openjdk7 package installed.
- You must have apache-ant installed.

Figure 1.

Figure 2.

First Steps

You need to go to the folder where you have extracted the freecol sources and then type ant. This will start building the freecol application with debugging information (you will see as GDB the -g flag is used by the compiler if you look at the build.xml). Wait as it will take a few minutes and you will have a new FreeCol.jar file.

How to Start Debugging

There are two ways to start debugging using JDB. The first one is just to give JDB the initial class (the one that has the main function) and starts from there as the JVM has not started. You must type run to start the program. The second one is what we will use. The second approach is to start a JVM and to connect to it:

```
$ java -jar -Xmx256M -Xdebug -Xrunjdwp:transport=dt_
   socket,server=y,address=6000 FreeCol.jar -no-intro
```

Here, I use the `-no-intro` parameter because there is currently an issue with freecol where the intro movie freezes the application. Then, we need to attach to this running JVM with the following command:

```
$ jdb -attach localhost:6000
```

Now, let's start with some basic commands.

Setting breakpoints

First thing you need to know is how to set breakpoints:

- `stop in <class-name>.<method-name>` Stop on entry to the given method.
- `stop at <class-name>:<line-number>` Stop at the given line.
- `clear <class-name>.<method-name>` Remove the specified breakpoint.

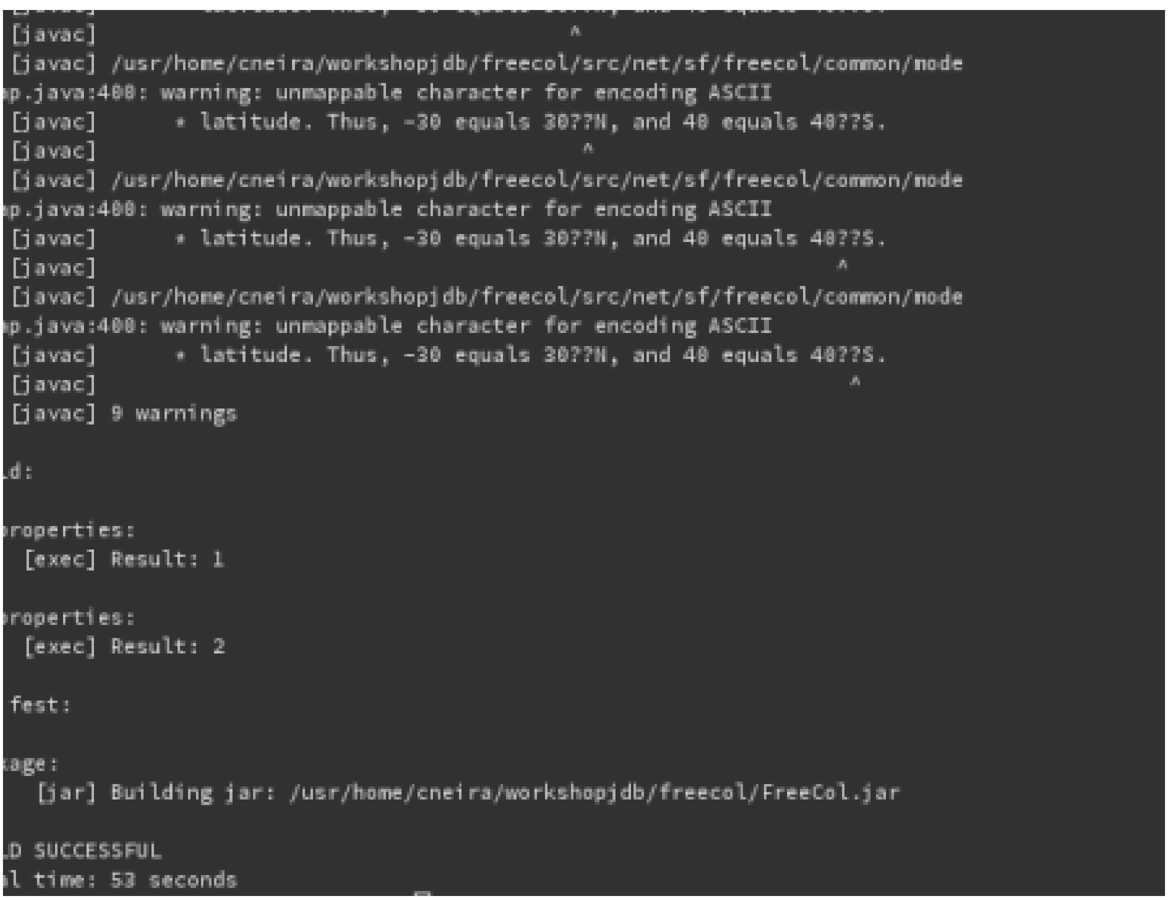


Figure 3.

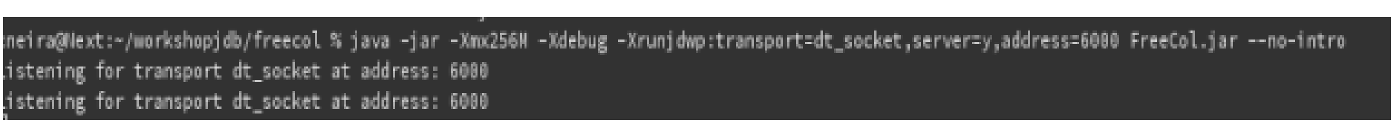


Figure 4.

- clear <class-name>:<line-number> Remove the specified breakpoint.

Stepping in Stepping out

When you hit a breakpoint, you either take a look at the data or continue executing the program. Here is the syntax for continuing, stepping, etc.

- cont – As the name says to just continue the execution after you hit the breakpoint
- step – Execute the current line. If the breakpoint is at a method call, it will stop the program at the method entry (same as GDB)
- next – Execute the current line. If the breakpoint is at a method call, do not stop at the method entry (same as GDB)
- step up – Execute until the current method returns to its caller (in GDB the command is called finish)

Take a look at the source code – As GDB this command has the same name you need to execute jdb where the source is located or use the use command in the JDB and provide the directory with the source code.

list – List 10 lines starting 4 before the current line.

- list <linenumber> – List 10 lines starting 4 before the given line.
- list <method_name> – List the first 10 lines of the given method.

Taking a peek

To look at the values of variables or expressions we have the following commands:

- print <name> – Print the current value of the given variable.
- print <expression> – Print the value of the given expression.
- locals – Print the values of all variables local to the current method (In GDB this one is called Info locals)

Calling for help

The same as GDB just type help.

Option flags

When you use the jdb command instead of the java command on the command line, the jdb command accepts many of the same options as the java command, including -D, -classpath, and -X options. The following list contains additional options that are accepted by the jdb command:

- help – displays a help message.
- sourcepath dir1:dir2: . . . – uses the specified path to search for source files in the specified path. If this option is not specified, then use the default path of dot (.).
- attach address – attaches the debugger to a running JVM with the default connection mechanism.
- listen address – waits for a running JVM to connect to the specified address with a standard connector.
- launch – starts the debugged application immediately upon startup of JDB. The -launch option removes the need for the run command. The debugged application is launched and then is stopped just before the initial application class is loaded. At that point, you can set any necessary breakpoints and use the cont command to continue execution.
- listconnectors – List the connectors available in this JVM.
- connect connector-name:name1=value1 – Connects to the target JVM with the named connector and listed argument values.
- dbgtrace [flags] – Prints information for debugging the jdb command.
- tclient – Runs the application in the Java HotSpot VM client.
- tserver – Runs the application in the Java HotSpot VM server.
- Joption – Passes option to the JVM, where option is one of the options described on the reference page for the Java application launcher. For example, -J-Xms48m sets the startup memory to 48 MB.

Conclusions

JDB like GDB is a very powerful tool that will help you find bugs in your own code, hopefully with little effort. I hope you found this article useful. If you want to learn more, I invite you to my workshop on Application Debugging and Troubleshooting.

CARLOS NEIRA

Carlos Neira has worked several years as a C/C++ developer and kernel porting and debugging enterprise legacy applications. He is currently employed as a C developer under Z/OS, debugging and troubleshooting legacy applications for a global financial company. Also he is engaged in independent research on affective computing. In his free time he contributes to the PC-BSD project and enjoys metal detecting.

Flaws in Unix-like Rootkits and Anti-rootkit Tools

Most high-profile intrusions of the past decade haven't been found due to intrusion detection sensors or complicated forensic tools. Instead, it was the malfunctioning code in the rootkits that made them crash the system with weird error messages.

Most of you already know that the rootkit is the basic set of tools an intruder uses to keep control over a compromised system. Common features of a rootkit include:

- Remote access
- Ability to intercept data
- Hiding modifications on the filesystem
- Hiding processes
- Hiding 'magic' users
- Hiding remote access ports/connections.

There are two types of rootkits, based on the way they interfere with user actions: userland and kernel land rootkits. The difference between them comes from the methods they use to hijack the operating system functions.

Rootkits work by subverting the normal operations of the system, either by modifying files on the system with backdoor versions (*userland rootkits*) or hijacking the operating system function/handler pointers in memory to alter the normal behavior (*kernel land rootkits*) in order to keep SUPER USER rights. In order to be able to keep control over a compromised system, a *userland rootkit* has to modify or alter a binary file in some way, either by replacing the file, modifying parts of the file on disk or modifying parts of the process memory space. Such modifications are, however, easily spotted by a trained admin eye because userland rootkits provide a poor means of disguising the presence of the attacker and his tools, a poor means of hiding the remote connection, not many options to survive reboot, etc.

```
# Adore Rootkit. OK, nobody calls it that but basically it uses Adore.
# In one commercial AV vendors naming scheme it's called Dextenea.
AKIT_FILES="${RKHROOTDIR}/usr/secure
            ${RKHROOTDIR}/usr/doc/sys/qrt
            ${RKHROOTDIR}/usr/doc/sys/run
            ${RKHROOTDIR}/usr/doc/sys/crond
            ${RKHROOTDIR}/usr/sbin/kfd
            ${RKHROOTDIR}/usr/doc/kern/var
            ${RKHROOTDIR}/usr/doc/kern/string.o
            ${RKHROOTDIR}/usr/doc/kern/ava
            ${RKHROOTDIR}/usr/doc/kern/adore.o
            ${RKHROOTDIR}/var/log/ssh/old"
AKIT_DIRS="${RKHROOTDIR}/lib/security/.config/ssh
           ${RKHROOTDIR}/usr/doc/kern
           ${RKHROOTDIR}/usr/doc/backup
           ${RKHROOTDIR}/usr/doc/backup/txt
           ${RKHROOTDIR}/lib/backup
           ${RKHROOTDIR}/lib/backup/txt
           ${RKHROOTDIR}/usr/doc/work
           ${RKHROOTDIR}/usr/doc/sys
           ${RKHROOTDIR}/var/log/ssh
           ${RKHROOTDIR}/usr/doc/.spool
           ${RKHROOTDIR}/usr/lib/kterm"
AKIT_KSYMS=
```

Figure 1. *Rootkit detection in RKHUNTER – ADORE*

On the other hand, *kernel land rootkits* need to interfere with binaries only during the booting phase, to make sure they are loaded during the system initialization procedure. Also, they provide an incredible means of hiding an attacker's presence inside a computer system, by manipulating the most low-level routines of the operating system. Also, they make remote access really easy for the attacker, with little to no warnings to the administrator. They are very hard to detect by rootkit detection tools.

```
# Adore Rootkit. OK, nobody calls it that but basically it uses Adore.
# In one commercial AV vendors naming scheme it's called Dextenea.
AKIT_FILES="${RKHROOTDIR}/usr/secure
            ${RKHROOTDIR}/usr/doc/sys/qrt
            ${RKHROOTDIR}/usr/doc/sys/run
            ${RKHROOTDIR}/usr/doc/sys/crond
            ${RKHROOTDIR}/usr/sbin/kfd
            ${RKHROOTDIR}/usr/doc/kern/var
            ${RKHROOTDIR}/usr/doc/kern/string.o
            ${RKHROOTDIR}/usr/doc/kern/ava
            ${RKHROOTDIR}/usr/doc/kern/adore.o
            ${RKHROOTDIR}/var/log/ssh/old"
AKIT_DIRS="${RKHROOTDIR}/lib/security/.config/ssh
           ${RKHROOTDIR}/usr/doc/kern
           ${RKHROOTDIR}/usr/doc/backup
           ${RKHROOTDIR}/usr/doc/backup/txt
           ${RKHROOTDIR}/lib/backup
           ${RKHROOTDIR}/lib/backup/txt
           ${RKHROOTDIR}/usr/doc/work
           ${RKHROOTDIR}/usr/doc/sys
           ${RKHROOTDIR}/var/log/ssh
           ${RKHROOTDIR}/usr/doc/.spool
           ${RKHROOTDIR}/usr/lib/kterm"
AKIT_KSYMS=
```

Figure 2. *Rootkit detection in RKHUNTER – SUCKIT*

```
# Phalanx Rootkit
PHALANX_FILES="${RKHROOTDIR}/uNFuNF
               ${RKHROOTDIR}/etc/host.ph1
               ${RKHROOTDIR}/bin/host.ph1
               ${RKHROOTDIR}/usr/share/.home.ph1/phalanx
               ${RKHROOTDIR}/usr/share/.home.ph1/cb
               ${RKHROOTDIR}/usr/share/.home.ph1/kebab"
PHALANX_DIRS="${RKHROOTDIR}/usr/share/.home.ph1
              ${RKHROOTDIR}/usr/share/.home.ph1/tty"
PHALANX_KSYMS=


# Phalanx2 Rootkit
PHALANX2_FILES="${RKHROOTDIR}/etc/khubd.p2/.p2rc
                ${RKHROOTDIR}/etc/khubd.p2/.phalanx2
                ${RKHROOTDIR}/etc/khubd.p2/.sniff
                ${RKHROOTDIR}/etc/khubd.p2/sshgrab.py
                ${RKHROOTDIR}/etc/lolzz.p2/.p2rc
                ${RKHROOTDIR}/etc/lolzz.p2/.phalanx2
                ${RKHROOTDIR}/etc/lolzz.p2/.sniff
                ${RKHROOTDIR}/etc/lolzz.p2/sshgrab.py
                ${RKHROOTDIR}/etc/cron.d/zupzzplaceholder
                ${RKHROOTDIR}/usr/lib/zupzz.p2/.p-2.3d
                ${RKHROOTDIR}/usr/lib/zupzz.p2/.p2rc"
PHALANX2_DIRS="${RKHROOTDIR}/etc/khubd.p2
               ${RKHROOTDIR}/etc/lolzz.p2
               ${RKHROOTDIR}/usr/lib/zupzz.p2"
PHALANX2_KSYMS=
```

Figure 3. *Rootkit detection in RKHUNTER – PHALANX*

However, the complexity of code may result in unexpected behavior.

So how do *anti-rootkit tools* work? They are founded on the principle that the rootkit needs to make certain changes to an operating system in order to work. *User land rootkits* will alter files on disk, timestamps, file sizes, the directory structure, etc. *Kernel land rootkits* alter system calls and functions, most of them focusing on the syscall table. Any type of rootkit will add files to the file system; maybe start new processes or new remote connections. For instance, the fact that rootkits keep the same file structure for their own files makes them easily spotted by traditional *anti rootkit tools* such chkrootkit or rkhunter because these tools use the directory and file structure of rootkits to generate signatures of malware presence (Figures 1-5).

They parse certain system directories looking for files known to be the part of malware, so when such a suspected file is found, the user is alerted.

However, as you can easily see, newer or slightly modified versions of rootkits go undetected by traditional signature scanning methods. Therefore, there is a need for a periodically updated database with the latest rootkit signatures and even then, there is no guarantee that a rootkit did not evade the signature scan.

The new technology has been developed in order to avoid relying on signatures or known-to-be-sane system fingerprints to scan for. Instead, it uses live system information it gathers to be able to heuristically determine signs of rootkit activity. Furthermore, we are able to uniquely pinpoint the rootkit code, thus being able to analyze it and determine its exact functionalities, and also extract a unique signature specific to the typology of the rootkit that can be used to speed up further scans, and also, to heuristically determine new or slightly modified versions of the rootkit.

So, how do newer rootkits get detected using traditional means? Well, they don't. Instead, rootkits make visible changes

to the operating system, sometimes resulting in system crashes and weird error messages, mostly due to insufficiently tested code. For example, Phalanx tries to disguise itself as Xnest to access /dev/mem without checking if there is Xnest on the system. An admin would be irritated seeing a message such as "Program Xnest tried to access /dev/mem between 0->8000000" when he doesn't even have Xnest running. And other kernel rootkits have their own distinctive flaws.

So basically, rootkits are found because they hijack predictable places, mostly aiming at the syscall table, Interrupt Descriptor Table or hijacking filesystem operations. Most modern anti-rootkit detection tools check all these places for inconsistencies with the values previously gathered in their database making visible such a modification. The major flaw of this approach is that if the system is already compromised, or if the attacker has access to the stored fingerprint, or fingerprint update mechanism, they can easily go unnoticed.

Another problem of a rootkit is that they usually tend to employ as many functionalities as possible. This leads to complex code which is a very dangerous bet, especially in Kernel land as it increases the number of possible points of failure. Also, most rootkits don't get a consistent grasp of the specifics of the system they run on.

Another major problem of a rootkit is the *remote access*, because an alert admin could easily spot unknown traffic going to suspicious ports, especially if it's encrypted or encapsulated. Eavesdropping the connection can provide important information about the intruder.

But as we have realized, current Unix anti-rootkit tools provide little to no accuracy in detection of rootkits, the impossibility to clean the system from a rootkit infection or the ability to analyze the malware. It looks like anti-rootkit tools have to step the notch up a bit and raise the bar a little higher for rootkit programmers, so that they come up with better and newer detection mechanisms instead of relying on known fingerprints or signatures, as these can easily be evaded or forged.

AB CONSULTANCY SOFTWARE SRL

Is a newly merged computer security company located in Bucharest, Romania whose main area of activity is penetration testing and forensics examination. Our experts have over 20 years of international experience in the field of computer security research, both offensive and defensive security, ranging from malware and antimalware research, software audit, exploit development or cryptology. Our customers range from government, military and financial industries, both based in Romania and abroad.

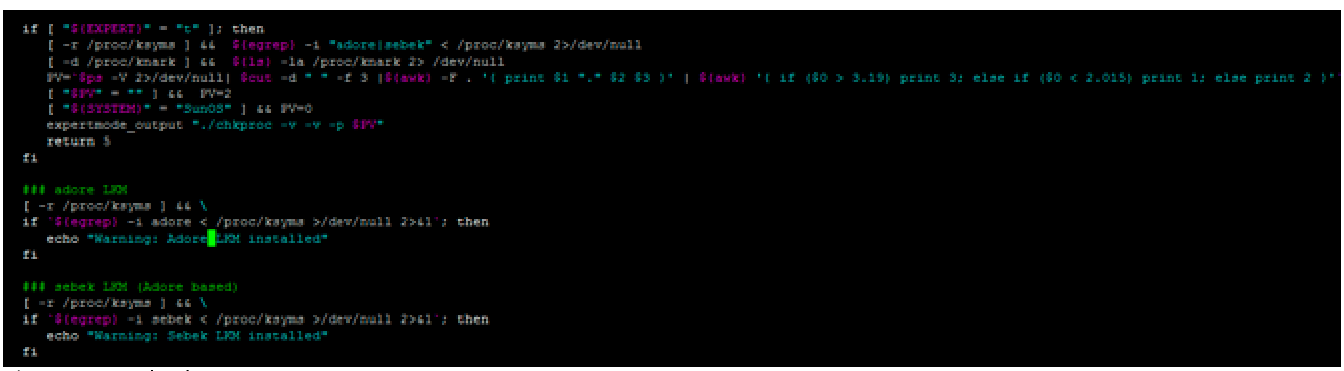
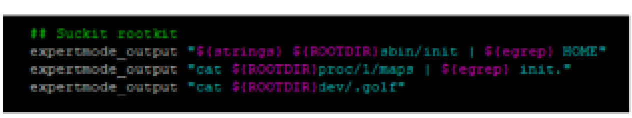


Figure 4. *Rootkit detection in CHKROOTKIT – ADORE*



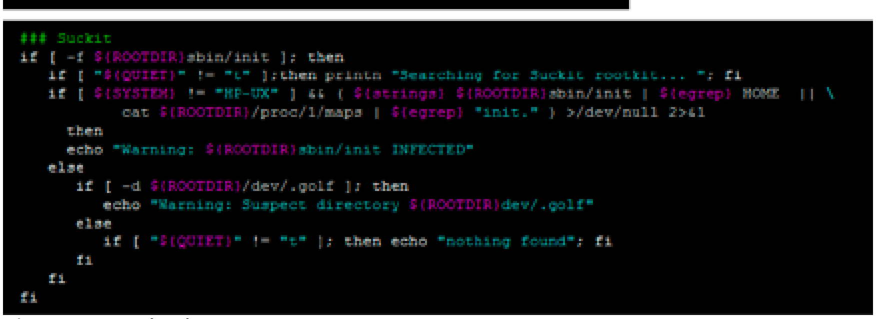


Figure 5. *Rootkit detection in CHKROOTKIT – SUCKIT*

100+ Unix Commands. Part 1

We will analyze and present the top 100 commands in Unix/ Linux that an attacker (and most crucially, a security tester) can use. These will range from commands to gain access to a system or extend access, to altering logs and other files (including the kernel) and to monitoring what is going on. We will also provide scripting techniques and examples based on these commands to provide the basics needed by any budding *NIX security professional.

System testing requires knowledge of many types of systems. In addition to Windows, the tester needs to understand the Linux and *NIX operating systems. Even in a Windows focused network, it is common to discover a *NIX system running an often overlooked but essential function that is critical to the organisation.

The designation *NIX is used throughout this article to designate either Linux or UNIX based systems. Although Linux is not actually UNIX, it is enough of a UNIX-like system that we can treat it the same for all non-semantic purposes.

For this and many other reasons, it is essential that any security tester has an understanding of both UNIX and Linux based systems. Not only will you come across these in the course of your career, but there are many excellent (and free) tools included in *NIX systems that will aid the coder and tester.

My goal is to provide a foundation into the most essential commands needed by the system tester. This includes both the system security tester and the penetration tester.

To this end, we will start with the simple commands used to navigate the *NIX system and work towards a more detailed approach associated with breaking into a system. This is not designed to teach people to be "hackers" or "crackers", but rather to aid people in testing the security of the operating system. Many of these skills do overlap; the motivation and means vary.

As a consequence, this article is not aimed at teaching people how to either secure a *NIX system or to effectively run it. It will address the fundamentals of using the system, but from the perspective of testing the security of that system; and more importantly from the perspective of attempting to bypass the controls in the system.

We will introduce the fundamental tools needed to create both shell exploits and other attacks as well as provide an introduction to basic code checking and compilation tools. None of these will make the reader an expert in the use of these tools and only a cursory coverage of these tools can be provided in an article of this format (a subsequent GREM paper on the creation of shell scripts and the advanced use of these tools is planned as a follow-up paper later).

To this end, this article begins by looking at a cursory high level overview of the most common *NIX tools and works toward a more in depth coverage of these and several more advanced tools. This will also include an increased use of scripting to incorporate several commands in order to extend their functionality.

The key to any *NIX system is to remember that you can highly customize the system. Any *NIX system is highly configurable and customizable, letting the tester do far more than any responsible system administrator would allow. The more the tester understands about the system and the range of functionalities of these commands, the more versatile and effective they will be.

To this end, we begin by looking at the most commonly issued commands and those that you will expect to use any time you access any *NIX system. The article will begin with a simple introduction to a number of the commands, and then will go into more and more depth with those commands in order to stimulate your imagination as to just how much can be done with a number of simple commands under *NIX.

Let the fun begin…

Basic UNIX commands

Many of these are not on all UNIX or Linux systems. When connecting to a system for the first time, it is always a good idea to verify the operating system version.

Next, use a command such as which to validate the existence and path of commands that you intend to use.

If a command is not in your path, the `which` command will not return any information about the command you are checking.

This does not mean that the command is not installed on the system; it is just that you do not have it in your path. Offline investigation into the system defaults (there are many good information sources on the Internet) for the particular operating system version is the first step in finding what should be on the system.

Remember that *NIX systems are often customised by the system administrator, so expect far more variation in path and commands than you will expect to find on a Windows based system. To this end, search tools, file listing tools and text editors will become extremely important in conducting an initial reconnaissance of the system.

Before anyone can master the fine arts of bypassing controls and running shell exploits, it is essential that the basics are not just understood, but mastered. There are a number of key areas and commands that are fundamental to *NIX. Failure to master these commands is a limitation to achieving success or being that person who would run 'dir' on a Solaris host when 'ls' is the appropriate command. Among the most fundamental command areas are listed below:

- Account management (useradd, passwd, su, sudo, login, who, whoami, export);
- File management (cd, pwd, ls, mount, mkdir, chmod, chgrp, cp, mv, del, find, locate, mcedit, cat, more/less, pg);
- Network management (nc, ifconfig, ifdown/ifup, ping, netstat, route, tcpdump, cat /dev/ip, cat /dev/tcp, ssh, ftp, tftp);
- Executing programs (PATH, which, './', ps, jobs, at, cron);
- File editing (vi, emacs, mail);
- Compression to Compilation (tar, gzip, gunzip, rpm, configure, make);
- Scripting and more (perl, C, shell scripts, sed/awk, grep, man, info, shutdown, kill, bg, etc).

When conducting a reconnaissance on a NIX system, it is a good practice to ensure that you have the latest version of the system. It is also important to record the time/date that it is set to. These objectives are achieved using the `date` and `uname` commands (detailed below).

date

The `date` command is used to either display (to STDOUT) or set the system date and time. Entering the command `date` by itself will display the date and time of the system. An example of the output of the "date" command is shown below:

Thu Dec 18 15:14:21 AEST 2008

The command, `date -s "12/22/2008 17:23:59"` would be used in order to set the system's date and time to that displayed within the command.

The date command tells you a lot about a system. From this one command, you can gain an understanding as to the level of care that is applied to the host, its location and other information. If the clock skew (the distance from the real time) is large, then the system is likely to receive less than adequate care. In addition, if the clock is inaccurate, it will be simpler to hide an attack in the system logs. The date command will also return the time zone that is configured on the system. You will not always know where a host is located, and this information can aid you in determining where the system resides.

It is necessary to have access to a privileged account (root) in order to be able to set the system date and time. This will be covered in more detail later in the paper.

uname

The `uname` command gives a wealth of information to anybody who does not have knowledge of a system. This command is equivalent to the `ver` command in Microsoft Windows systems (and DOS). This single command provides intelligence as to the following:

- The Operating System (O/S) running on the host,
- The O/S kernel name and version (which can offer information as to patching practices associated with the host,
- The hosts processor type (e.g. i386, i686, sparc, mips) and the hardware platform (e.g. Sun-Fire-280R), and
- The O/S or kernel release level.

The following example of this command displays a wealth of information returned from this command.

```
$uname -a
=> Linux linux-0915 2.6.25.16-0.1-pae #1 SMP 2008-08-21
   00:34:25 +0200 i686 i686 i386 GNU/Linux
```

In some *NIX variants (this includes AT&T UNIX System V Release 3.0 – such as Solaris), the `setname` command is available and can be used to modify the values that are returned from the `uname` command. As such, it is not possible to trust all of the information that these initial reconnaissance commands return. It is a good start.

which

The `which` command is used to find the location (path) of a program file if it exists in the user's path. If the command being sought is not in the user path but is on the system, the `which` command will not return anything.

Table 1. *Terminal key inputs*

Key Sequence	Result
CTRL-j	Line feed. "CTRL-J" can act as a reset command on some systems.
CTRL-u	Remove the current line from the command buffer (and do not log it).
CTRL-s	Stop (and restart) a process. This sequence may suspend a command (i.e. pause it). CTRL-s stops the system from sending any further data to the screen until a CTRL-q is pressed.
CTRL-s	CTRL-s and CTRL-q control the flow of output to the terminal. CTRL-q resumes a terminal following the CTRL-s key.
CTRL-z	Use CTRL-Z to stop a job. The command "bg" (background) can be used to put the program into the background (type „bg" on the command line).
CTRL-c	Interrupt a program. This is used to "break" a program execution. This signal allows the program to clean up before exiting.
CTRL-d	This is an end-of-input response. Some commands (such as mail) require that you run this command. Use the command with caution as it may log you out of some shells.
CTRL-\	Kills a program (without cleaning up).

The `which` command prints the full path of the executable (if found) to stdout. This is done by searching through the directories listed in the environment variable PATH belonging to the user. The process updates the access times of the files and directories searched.

General Terminal use

When using a command terminal (command line) in *NIX, remember that it is simple to correct an error when typing a command. This can be done by typing "CTRL-u". Doing this will discard the entire line, removing any input from the buffer (and hence evidence such as the BASH history file). Unlike Windows, *NIX operating systems are case-sensitive. It matters what the case you use is.

*NIX allows commands to be put into "sleep", killed, or sent into the background. Some of these commands are discussed in Table 1. The table includes a list of keys that can be used in a shell and are listed with their effect.

In addition, there are a number of other simple techniques for controlling execution in a *NIX terminal shell. These include sending output to the foreground (fg), background (bg or &), redirecting it (> or >>) or piping it to another command (|).

The Essential Commands

The following list provides a quick introduction to the essential commands that you will need to navigate the *NIX system.

Ls Lists the files in a directory

The `ls` command is used to list the files in a directory. It is equivalent to the Microsoft 'dir' command, but far more powerful. As all hardware, memory etc are treated as a file in *NIX, any device can be accessed as a file – if you know where it is and have the correct permissions.

The `ls` command can be pointed at a single file, group of files, the current directory or a specific directory elsewhere in the tree.

ls –l

This command lists file entries using the 'long format'. It displays more detailed information compared to the 'ls' command. The information displayed includes file permissions, the size of the file, the file owner and group and the time when the file was last modified. The last modified time is important to note as a change may quickly alert a vigilant system administrator to a change.

ls -a

This command will list all files – even the hidden ones. In *NIX, a file is "hidden" similar to a Windows hidden file attribute through having a name that starts with a "." or a full-stop.

ls -r

The "r" flag instructs the 'ls' command to display its output in reverse order.

ls –t
The “t” flag instructs the `ls` command to display its output in order of the file timestamp. This allows you to quickly find all files that have been changed in a selected period of time.

Used together, these options can help you find all of the files in a directory that have been changed within the time that you have been logged into a host. For example, the command combination, `ls -altr | pg` will output all the files in the current directory in the order of timestamps starting with the most recently altered or added files and working to the oldest. Further, by piping the `pg` (page) command to `ls`, you can see the output of a single screen (page) at a time (rather than having this scroll past you faster than you can read it).

File Commands

The following is an introduction to a number of *NIX commands that are used to display or manipulate files.

more <filename> – less <filename> –
The `more` command displays a file one page at a time. It is similar to the ‘pg’ and ‘less’ commands, with the added benefit that it is available on practically any *NIX system.

The `pg` and `less` commands may be more powerful, but they are not universally available. Hitting the space bar will scroll to the next page and hitting `q` will quit the command. The option `/pattern`’is available on most *NIX systems and provides the ability to search for a defined pattern within the file.

emacs <filename> – vi <filename>
Both `emacs` and `vi` are text editors. However, people have a clear preference for one over the other. Of the two, `emacs` is far more powerful compared to `vi`. Both of these commands are used to create or edit a file. It is essential that you become adept in the use of the two commands. As `vi` is available on nearly every *NIX* system ever built – even most firewalls, it is essential to understand how to use it.

Although `emacs` is more powerful in the options it provides, it is commonly removed from many secured systems (such as IDS, gateways and Firewalls). With the knowledge of using both of these commands, it is unlikely that you will find yourself without access to a text editor when you need one.

mv <filename-1> <filename-2>
The `mv` command simply moves a file. It either renames the file to use a different name in the same directory (that is, it does not create a copy) or moves it into a different directory.

cp <filename-1> <filename-2>
The `cp` (copy) command copies a file. It is similar to what the `mv` command does with the only difference being that the original file will remain unchanged (only the file access timestamp will be updated).

rm <filename>
The `rm` command removes or deletes a file. It is equivalent to the Windows `del` command. The `-i` option of the command asks for confirmation. That is, `rm -i` will require you to confirm that you actually wish to delete the file before it actually removes it.

Many system administrators will alter their `.cshrc` file so that the default behaviour of the `rm` command is to ask for a confirmation. This can be problematic if you are creating scripts based on the user’s profile.

diff <filename-1> <filename-2>
The `diff` command compares two files and displays the differences. It is used to find changes to a file.

wc <filename>
The `wc` (word count) command displays the number of lines, words, and characters in a file.

chmod options <filename>
This command will be covered in detail later in the paper. The `chmod` command is used to change the permissions on a *NIX file. These permissions include the ability to read, write, and execute a file.

Without the correct permissions, you will not be able to even look at a file, alter it or run it.

File Compression and Concatenation

It is simpler to get files onto or off of a system as a group. Next, it is quicker (and there will be a lower footprint) in copying a smaller file. This is where compression and concatenation come into effect. There are many compression utilities that are available, but again, I have stuck to the simplest and most commonly available.

tar <filename>
The `tar` command creates a “tarball”. This is a concatenation of many files into a single file (that was originally used for a tar archive or backup). The command is also used to extract files. The `-c` option creates a new archive whereas the `-x` option extracts files (that is, restores them).

The command can be used on any files or directories. The `tar` command can be used against an entire subtree. The command options need not be preceded by the ‘-‘ character (although this is good practice). The long-style

options (those that use the '--' format/'syntax still require full definition).

Tarballs can be added too (this is done using the '-A' option) or simply listed (the '-t' option).

This command is far more complex than it first seems and it is essential that time be spent reading and comprehending the full extent of the command ('man tar' is a start).

gzip <filename>

This command is used to compress a file or a number of files. This reduces the time to copy a group of files over the network and makes a tarball less obvious. Other tools (such as 'compress' also exist but are less commonly used). The 'gzip' algorithm has one of the highest compression rates of any of the compression commands used in *NIX.

By default, the `gzip` command is used to create compressed files that use the extension '.gz'. The extension '.tgz' generally refers to a compressed tarball.

gunzip <filename>

The `gunzip` command uncompresses a file that has been compressed using the 'gzip' command (above).

gzcat <filename>

The `gzcat` command is far less common than the 'gzip' command. Although it may not be found on a target system, you should become familiar with this command as it allows the user to view the contents of a file that has been compressed using 'gzip' without decompressing it.

In effect, this command is analogous to `gunzip -c` with a simple ability to redirect or print the output (for instance the command `gzcat <filename> | lpr` will print the contents of the compressed file called `<filename>`).

Printer Control

Many *NIX systems have printer daemons even when they do not have an attached printer. At other times, printers are poorly configured. For this reason, it is important to understand the range of printer commands available when testing a *NIX system.

lpr <filename>

The `lpr` command is the main *NIX print command used to essentially print a file. The '-P' option is used in order to select a particular printer to send the print job to (that is, other than the default printer if one has been configured).

Many vulnerabilities have occurred as a result of poorly configured print queues. Consequently, learning these commands becomes essential when testing systems where the LP (line printer) daemon has been configured. More importantly, most *NIX systems have the LP daemon running by default. In addition to this, important information that can be used by the system tester can be commonly discovered when careless system administrators print sensitive information using badly configured printer queues.

lpq

The `lpq` command is used to view or display the printer queue. This can be used to obtain a job number to delete a print job, or to redirect and view it.

lprm <print_job_number>

This command is used to delete a job from the printer queue. It is good practice to specify the printer name or the command will default to the default printer (if one has been configured on the system).

Directory Commands

Directories are used to group files together in a hierarchical structure.

mkdir <directory_name>

This command creates a new directory

cd <directory_name>

The `cd` command is used to change your location into a different directory. This allows you to display the file information in the directory with the `ls` command. When a user logs into a *NIX system, they will begin in their 'home directory'.

Returning to one's home directory is as simple as entering the `cd` command without any other argument or options. The command 'cd ..' moves you one level higher in the directory hierarchy towards the root or '/' directory.

pwd

Displays the current directory that you are in

ff

The `ff` command is used to find files on a *NIX host. The `ff -p` command does not need the full name of the command being searched for. This command is similar to the command 'find' which is discussed in detail later.

grep <string_to_search> <filename>

The `grep` command extracts, displays or simply finds strings within a file. The command performs the searches based on regular expressions. There are a number of updated 'grep' programs including egrep and fgrep.

Finding out about other users on the system

These commands are used to find out about other users on a *NIX host. When testing the security of a system covertly

(such as when engaged in a penetration test), it is best to stop running commands when the system administrator is watching.

w

The `w` command displays any user logged into the host and their activity. The command is used to determine if a user is 'idle' or if they are actively monitoring the system.

who

The `who` command is used to find which users are logged into the host as well as to display their source address and how they are accessing the host. The command will display if a user is logged into a local tty (more on this later) or is connected over a remote network connection.

finger <user_name>

The `finger` command is rarely used these days (but does come up from time to time on legacy and poorly configured systems). The command provides copious amounts of data about the user who is being "fingered". This information includes the last time the user read their mail and any log in details.

last -1 <user_name>

The `last` command can be used to display the "last" user to have logged on and off the host and their location (remote or local tty). The command will display a log of all recorded logins and log-offs if no options are used. When the `<user_name>` option is provided, the command will display all of the user's log-ins to the system. This information is used when profiling a system administrator to discover the usual times that a person will be logged into and monitoring a system.

whoami

This command displays the username that is currently logged into the shell or terminal session.

passwd <user_name>

The `passwd` command is used to change your password (not options) or that of another user (if you have permissions to do this).

kill PID

This command "kills" any processes with the PID (process ID) given as an option. The 'ps' command (detailed later in the paper) is used to find the PID of a process.

This command can be used to stop a monitoring or other security process when testing a system. The `root` user can stop any process, but other users on a host can only stop their own (or their groups') processes by default.

du <filename>

The `du` command displays the disk usage (that is the space used on the drive) associated with the files and directories listed in the `<filename>` command option.

df

The `df` command is used to display the amount of free and used disk space on the system. This command displays this information for each mounted volume of the host.

Network Commands

rlogin <hostname>

A command to connect to a remote host and gain a shell based on trust connections (`.rhost` files).

rcp <hostname>

A command to run a command on a remote host

telnet <hostname>

A command to connect to a remote host and gain a shell using clear text authentication and data transmission

ftp <hostname>

The `ftp` command is used to upload or download files to/from a remote host running as an ftp server.

SSH

Secure shell (ssh) is in effect a remote shell (ssh) and remote file transfer program (scp). It is similar to telnet, but all data sent using ssh is encrypted.

"man" the *NIX online Manual

The `man` command may be used to view information or help files concerning the majority of *NIX commands. It is possible to conduct keyword searches if you are unsure of a command for a particular type of *NIX. Keyword search in "man" is provided by "apropos".

ldd

The command, `ldd` may be used to list dynamic dependencies of executable files or shared objects.

The `ldd` command can also be used to examine shared libraries themselves, in order to follow a chain of shared library dependencies.

The `pvs` command may also be useful. It displays the internal version information of dynamic objects within an ELF file. Commonly, these files are dynamic executables and shared objects, and possibly reloadable objects.

This version information can fall into one of the following two categories:

- Version definitions describe the interface made available by an ELF file. Each version definition is associated to a set of global symbols provided by the file.
- Version dependencies describe the binding requirements of dynamic objects on the version definition of any shared object dependencies. When a dynamic object is built with a shared object, the link-editor records information within the dynamic object, indicating that the shared object is a dependency.

For example, the command `pvs -d /usr/lib/libelf.so.1` can be used to display the version definition of the ELF file libelf.so.1.

*NIX commands for file permissions

"Chmod"

The `chmod` command is used to modify permissions on a file or directory. It supports both character notation (e.g. `chmod o+x file`) and octal notation (as discussed above).

"cat" or Concatenate

The `cat` command is similar to the `type` command on Microsoft windows. This command is generally used to output or view the contents of a file. The Command can also be used to join or concatenate multiple files together.

Authentication and Validation

There are a variety of ways in which a user can be authenticated in UNIX. The two primary differences involve authentication to the operating system against authentication to an application alone. In the case of an application such as a window manager (e.g. X-Window), authentication to the application is in fact authenticating to the operating system itself. Additionally, authentication may be divided into both local and networked authentication. In either case, the same applications may provide access to either the local or remote system. For instance, X-Window may be used both as a local window manager and as a means of accessing a remote UNIX system. Additionally, network access tools such as SSH provide the capability of connecting to a remote host but may also connect to the local machine by connecting to either its advertised IP address or the local host (127.0.0.1) address.

The UNIX authentication scheme is based on the `/etc /passwd` file. PAM (pluggable authentication modules) has extended this functionality and allowed for the integration of many other authentication schemes. Pam was first proposed by Sun Microsystems in 1995 and was integrated into Red Hat Linux the following year. Subsequently, PAM has become the mainstay authentication schema for Linux and many UNIX varieties. PAM has been standardized as a component of the X/Open UNIX standardization process.

This resulted in the X/Open Single Sign-on (XSSO) standard. From the assessor's perspective, PAM however, necessitates a recovery mechanism that needs to be integrated into the operating system in case a difficulty develops in the linker or shared libraries. The assessor also needs to come to an understanding of the complete authentication and authorization methodology deployed on the system. PAM allows for single sign-on across multiple servers. Additionally, there is a large number of plug-ins to PAM that vary in their strength. It is important to assess the overall level of security provided by these and remember that the system is only as secure as the weakest link.

The fallback authentication method for any UNIX system lies with the `/etc/passwd` (password) file. In modern UNIX systems, this will be coupled with a shadow file. The password file contains information about the user, the user ID (UID), the group ID (GID), a descriptor which is generally taken by the name, the user's home directory and the user's default shell (see Figure 1).

The user ID and group ID give the system the information needed to match access requirements.

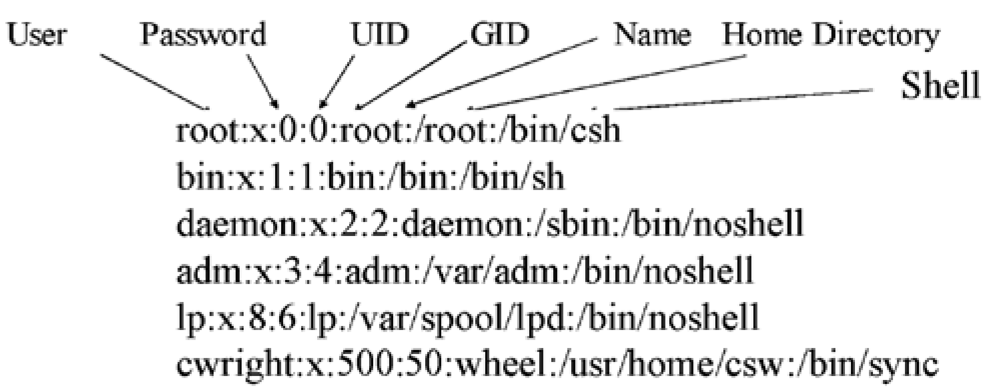


Figure 1. *The /etc/passwd File*

The home directory in the password file is the default directory that a user will be sent to in the case of an interactive login. The shell directive sets the initial shell assigned to the user on login. In many cases, a user will be able to change directories or initiate an alternative shell, but this at least sets the initial environment. It is important to remember that the password file is generally world readable. In order to correlate user IDs to user names when looking at directory listings and process listings, the system requires that the password file be accessible to all (at least in read only mode) authenticated users.

The password field of the `/etc/passwd` file has a historical origin. Before the password and show files were split, hashes would be stored in this file. To maintain compatibility, the same format has been used. In modern systems where the password and shadow files are split, an "x" is used to show that the system has stored the password hashes in an alternative file. If there is a blank space instead of the "x", this indicates that the account has no password.

The default shell may be a standard interactive shell, a custom script or application designed to limit the functionality of the user or even a false shell designed to restrict the use and stop interactive logins. False shells are generally used in the case of service accounts. This allows the account to login (such as in the case of "lp" for print services) and complete the task it is assigned. Additionally, users may be configured to run an application. A custom script could be configured to start the application allowing the user limited access to the system and to then log the user off the system when they exit the application. It is important for the assessor to check that breakpoints cannot be set allowing an interactive shell. Further, in the case of the application access, it is important to check that the application does not allow the user to spawn an interactive shell if this is not desired. Either of these flaws may be exploited to gain deeper access to a system.

As was mentioned above, the majority of modern *NIX systems deploy a shadow file. This file is associated with the password file but unlike the password file, it should not be accessible (even for reading) by the majority of users on the system. The format of this file is:

User>Password_Hash>Last_Changed>Password Policy

This allows the system to match the user and other information in the shadow file to the password file.

The password is actually a password hash. The reason that this should be protected comes from the reason that the file first came into existence. In the early versions of UNIX, there was no shadow file. Since the password file was world readable, a common attack was to copy the password file and use a dictionary to "crack" the password hashes. By splitting the password and shadow file, the password hash is not available to all users and thus, it makes it more difficult for a user to attack the system. The password hash function always creates the same number of characters (this may vary from system to system based on the algorithm deployed, such as MD5, DES etc.).

*NIX systems are characteristically configured to allow zero days between changes and 99,999 days between changes. In effect, this means that the password policies are ineffective. The fields that exist in the shadow file are detailed below:

- The username,
- The password Hash,
- The Number of days since 01 Jan 1970 that the password was last changed,
- The Number of days that must pass before the password can be changed,
- The Number of days after which the password must be changed,
- The Number of days before expiration that a user is warned,
- The Number of days after expiration that an account is disabled,
- The Number of days since 01 Jan 1970 that an account has been disabled.

Since the hash function will always create a password hash of the same length, it is possible to restrict logins by changing the password hash variable in the shadow file. For instance, changing the password hash field to something like `No _ login` will create a disabled account. As this string is less than the length of the password hash, no password hash could ever be created matching that string. So, in this instance, we have created an account that is not disabled but will not allow interactive logins.

Many systems also support complex password policies. This information is generally stored in the `password policy` section of the show file. The password policy generally consists of the minimum password age, maximum password age, expiration warning timer, post expiration disable timer, and a count for how many days an account has been disabled. Most system administrators do not know how to interpret the shadow file. As an auditor, knowledge of this information will be valuable. Not only will it allow you to validate password policy information, but it may also help in displaying a level of technical knowledge.

When assessing access rights, it is important to look at both how the user logs in and where they log in from.

Always consider the question of whether users should be able to log in to the root account directly. Should they be able to do this across the network? Should they authenticate to the system first and then re-authenticate as root (using a tool such as `su` or `SUDO`)? When auditing the system, these are some of the questions that you need to consider.

Many UNIX systems control this type of access using the `/etc/securetty` file. This file includes an inventory of all of the `ttys` used by the system. When auditing the system, it is important that you first collate a list of all locations that would be considered secure enough to sanction the root user to log into from these points. When testing the system, verify that only terminals that are physically connected to the server can log into the system as root. Generally, this means that there is either a serial connection to a secure management server or more likely, it means allowing connections only from the root console itself. It is also important to note that many services such as SSH have their own configuration files which allow or restrict authentication from root users. It is important to check not only the `/etc/securetty` file but any other related configurations files associated with individual applications.

Side note

TTY stands for teletype. Back in the early days of UNIX, one of the standard ways of accessing a terminal was via the teletype service. Although this is one of the many technologies that have faded into obscurity, UNIX was first created in the 1960s and 70s. Many of the terms have come down from those old days.

Usernames, UIDS, the Superuser

Root is almost always connected with the global privilege level. In some extraordinary cases (such as special UNIX'es running Mandatory Access Controls), this is not true, but these are rare. The super-user or `root` account (designated universally as UID "0") includes the capacity to do practically anything on a UNIX system. RBAC (role-based access control) can be implemented to provide for the delegation of administrative tasks (and tools such as `SUDO` or super-user do also provide this capability). RBAC provides the ability to create roles. Roles, if configured correctly, greatly limit the need to use the root user privilege. RBAC both limits the use of the `su` command and the number of users who have access to the root account. Tools such as SUDO successfully provide similar types of control, but RBAC is more granular than tools such as SUDO allowing for a far greater number of roles on any individual server. It will come down to the individual situation within any organization as to which particular solution is best.

File System Access Control

UNIX file level access controls are both simple and complex. Granting permissions to individual users or in small groups is simple. Difficulties may arise in cases where a system has to provide access to a large number of users or groups. In this situation, it is possible for groups to grow in number exponentially. UNIX file permissions are defined for:

- Owner
- Group
- World

The owner relates to an individual user. Restrictions on the owner associate file access with an individual. Group access provides the ability to set access restrictions across user groups. UNIX provides a group file (usually `/etc/group`) that contains a list of group memberships. Alternative applications have been developed for larger systems due to the difficulties associated with maintaining large numbers of group associations in a flat file database. The world designation is in effect equivalent to the Windows notion of everybody.

UNIX has three main permissions, read, write and execute. In addition, there are a number of special permissions that we will discuss below. The read permission provides the capability to read a file or list the contents of a directory. The write permission provides the capability to edit a file, or add or delete a directory entry. The execute permission provides the capability to execute or run an executable file.

UNIX also provides for a special capability with the setting of a "sticky bit". The "sticky bit" protects the files within a public directory that users are required to write to (e.g. the `/tmp` directory). This protection is provided through stopping users from having the capability to delete files that belong to other users which have been created in this public directory. In directories where the "sticky bit" has been set, only the owner of the file, owner of the directory, or the root user has the permissions to delete a file.

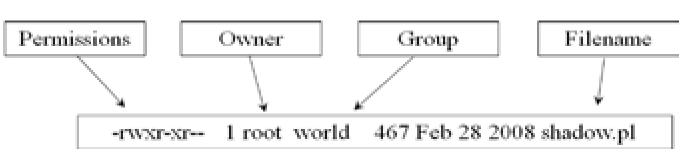


Figure 2. *Unix File Permissions*

The Unix file permissions are: "r, w, x, t, s, S". The following table demonstrates the octal format for "r" or read, "w" or write, and "x" or execute.

--x – execute
-w- – write
-wx – write and execute
r-- – read
r-x – read and execute
rw- – read and write
rwx – read, write and execute

The first character listed when using symbolic notations to display the file attributes (such as from the output of the `ls -l` command) indicates the file type:

- \- denotes a regular file
- b denotes a block special file
- c denotes a character special file
- d denotes a directory
- l denotes a symbolic link
- p denotes a named pipe
- s denotes a domain socket

The three additional permissions mentioned above are indicated by changing one of the three `execute` attributes (this is the execute attribute for user, group or world). The following table details the various special setuid and setgid permissions. There is a difference between whether the file special permission is set on an executable or non-executable file.

Permission	Class	Executable files	Non-executable files
Set User ID (setuid)	User	S	S
Set Group ID (setgid)	Group	s	S
Sticky bit	World	t	T

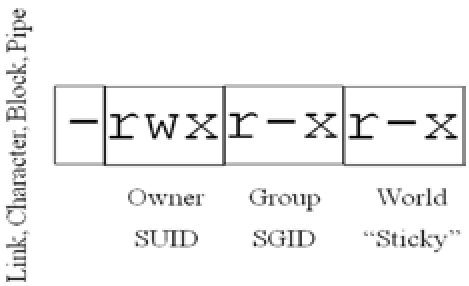


Figure 3. *Unix File Permissions (2)*

The following examples provide an insight into symbolic notation:

- -rwx r-x r--: this permission is associated with a regular file whose user class or owner has full permissions to run, read and write the file. The group has the permissions to read and execute the file. And the world or everyone on the system is allowed to only read the file.
- crw-r--r--: the symbolic notation here is associated with a character special file whose user or owner class has both the read and write permissions. The other classes (group and world) only have the read permission.
- dr-x------: this symbolic notation is associated with a directory whose user or owner class has read and execute permissions. The group and world classes have no permissions.

User level access

The UNIX file system commonly distinguishes three classifications of users:

- Root (or as the account is also called super-user),
- Users with some privilege level, and
- All other users.

The previous section on access controls showed us how UNIX privileges and the access to files may be granted with access control lists (ACLs). The simplicity of the UNIX privilege system can make it extremely difficult to configure privileges in UNIX. Conversely, it also makes them relatively simple to audit. The UNIX directory command, `ls -al` supplies the means to list all files and their attributes. The biggest advantage for an auditor is the capability to use scripting to capture the same information without having to actually visit the host. A baseline audit process may be created using tailored scripts that the audit team can save to a CD or DVD with statically linked binaries. Each time there is a requirement for an audit, the same process can be run. The benefits of this method are twofold. First, subsequent audits require less effort. Next, results of the audit can be compared over time. The initial order can be construed as a baseline and the results compared to future audits to both verify the integrity of the system and to monitor improvements. A further benefit of this method is that a comparison may be run from the tools on the system against the results derived from the tools on the disk.

Generally, it would be expected that no variation would result from the execution of either version of the tools. In the event that a Trojan or root kit found its way onto the server, the addition of a simple `diff` command would be invaluable. In the event that the diff command returned no output, it would be likely that no Trojan was on the system (except the kernel and lower level software). If on the other hand there was a variation in the results, one would instantly know that something was wrong with the system.

The primary benefit of any audit control that may be scripted is that it also may be automated. The creation of such a script and the association for a predetermined configuration file for the script economizes the auditors of valuable time allowing them to cover a wider range of systems and provide a more effective service. The selection of what to audit for on a file system will vary from site to site. There are a number of common configuration files associated with each version of UNIX and also a number of files and directories common to any organization. The integration of best practice tools such as those provided (see the Appendixes for further details) by the Centre for Internet Security, SANS, NIST and the US Department of Defense provide a suitable baseline for the creation of an individual system audit checklist.

Special permissions are set for a file or directory on the whole, not by a class.

The set user ID, setuid, or SUID permission

When a file for which this permission has been set is executed, the resulting process will presuppose the effective user ID given to the user class.

The set group ID, setgid, or SGID permission

When a file for which this permission has been set is executed, the resulting process will presuppose the group ID given to the group class. When setgid is applied to a directory, new files and directories created under that directory will inherit the group from that directory. The default behavior is to use the primary group of the effective user when setting the group of new files and directories.

The sticky permission

The characteristic behavior of the sticky bit on executable files allows the kernel to preserve the resulting process image beyond termination. When this is set on a directory, the sticky permission stops users from renaming, moving or deleting contained files owned by users other than themselves, even if they have write permission to the directory. Only the directory owner and superuser are exempt from this.

Blocking Accounts, Expiration, etc

The password properties in the `/etc/shadow` file contain information about password expiration in a number of fields. The exact method for enabling or disabling account expiration will vary between different *NIX varieties and the system security tester needs to become familiar with the system that they are to test. As was mentioned above, there are a number of ways to restrict access to accounts. It is necessary to create a check list that takes into account the individual differences that occurred across sites.

The `change` command changes the number of days between password changes and the date of the last password change. Information contained within the shadow file is used by the system to determine when the user must change their password. This information is valuable to the system security tester as well and the list of current users with the times that they last accessed the system and changed their passwords is an essential part of any *NIX security test or Penetration test. The command `change -l vivek` (which again will vary slightly by system) may be used to list currently ageing passwords on the existing accounts of the system. An example of the output provided from this command is shown below.

Last password change: Mar 20, 2008
Password expires: May 20, 2008
Password inactive: never
Account expires: never
Minimum number of days between password change: 3
Maximum number of days between password change: 90
Number of days of warning before password expires: 14

Restricting Superuser Access

Root access is rarely completely restricted to secured terminals – this leaves a backdoor in most systems. As was noted above, there are a variety of options and applications for which root may access the system. The security tester needs to gain a level of familiarity with the system such that they can check all logical routes that may be used to authenticate and authorize the user. Some of the more common avenues used to authenticate users on UNIX systems include:

- telnet (and other TTY based methods)
- SSH
- X-Window
- Local terminals and serial connections.

Finer Points of Find

The *NIX `find` command is probably one of the system security tester's best friends on any *NIX system. This command allows the system security tester to process a set of files and/or directories in a file subtree. In particular, the command has the capability to perform a search based on the following parameters:

- Where to search (which pathname and the subtree)
- What category of file to search for (use "-type" to select directories, data files, links)
- How to process the files (use "-exec" to run a process against a selected file)

- The name of the file(s) (the "-name" parameter)
- Perform logical operations on selections (the "-o" and "-a" parameters)

One of the key problems associated with the `find` command is that it can be difficult to use. Many experienced professionals with years of hands-on experience on *NIX systems still find this command a bit tricky. Adding to this confusion are the differences between *NIX operating systems. The find command provides a complex subtree traversal capability. This includes the ability to traverse excluded directory tree branches and also to select files and directories with regular expressions. As such, the specific types of file system searched with his command may be selected.

The find utility is designed for the purpose of searching files using directory information. This is in effect also the purpose of the `ls` command but find goes further. This is where the difficulty comes into play. Find is not a typical *NIX command with a large number of parameters, but is rather a miniature language in its own right.

The first option in find consists of setting the starting point or subtrees under which the find process will search. Unlike many commands, find allows multiple points to be set and reads each initial option before the first "-" character. That is, the one command may be used to search multiple directories on a single search. The paper, "Advanced techniques for using the *NIX find command" by B. Zimmerly provides an ideal introduction into the more advanced features of this command and is highly recommended that any system security tester become familiar with this. This section of the chapter is based on much of his work.

The complete language of find is extremely detailed consisting of numerous separate predicates and options. GNU find is a superset of the POSIX version and actually contains an even more detailed language structure. This difference will only be used within complex scripts as it is highly unlikely that this level of complexity would be effectively used interactively:

- -name: True if pattern matches the current file name. Simple regex (shell regex) may be used. A backslash (\) is used as an escape character within the pattern. The pattern should be escaped or quoted. If you need to include parts of the path in the pattern in GNU find, you should use predicate "wholename"
- "-(a,c,m)time" as possible search may file is last "access time", "file status" and "modification time", measured in days or minutes. This is done using the time interval in parameters -ctime, -mtime and -atime. These values are either positive or negative integers.
- -fstype type True if the filesystem to which the file belongs is of type type. For example, on Solaris, mounted local filesystems have type ufs (Solaris 10 added zfs). For AIX, local filesystem is jfs or jfs2 (journalled file system). If you want to traverse NFS filesystems, you can use nfs (network file system). If you want to avoid traversing network and special filesystems, you should use predicate local and in certain circumstances mount
- "-local": This option is true where the file system type is not a remote file system type.
- "-mount": This option restricts the search to the file system containing the directory specified. The option does not list mount points to other file systems.
- "-newer/-anewer/-cnewer baseline": The time of modification, access time or creation time are compared with the same timestamp in the file used as a baseline.
- "-perm permissions": Locates files with certain permission settings. This is an important command to use when searching for world-writable files or SUID files.
- "-regex regex": The GNU version of find allows for file name matches using regular expressions. This is a match on the whole pathname not a filename. The "-iregex" option provides the means to ignore case.
- "-user": This option locates files that have specified ownership. The option `-nouser` locates files without ownership. In the case where there is no user in `/etc/passwd`, this search option will find matches to a file's numeric user ID (UID). Files are often created in this way when extracted from a tar archive.
- "-group": This option locates files that are owned by a specified group. The option, "-nogroup" is used to refer to searches where the desired result relates to no group that matches the file's numeric group ID (GID) of the file
- "-xattr": This is a logical function that returns true if the file has extended attributes.
- "-xdev ": Same as the parameter "-mount primary". This option prevents the find command from traversing a file system different from the one specified by the Path parameter.
- "-size": This parameter is used to search for files with a specified size. The "-size" attribute allows the creation of a search that can specify how large (or small) the files should be to match. You can specify your size in kilobytes and optionally also use + or – to specify size greater than or less than the specified argument. For instance:
  - find /usr/home -name "*.txt" -size 4096k
  - find /export/home -name "*.html" -size +100k
  - find /usr/home -name "*.gif" -size -100k
- "-ls": list current file in "ls –dlis" format on standard output.

- "-type": Locates a certain type of file. The most typical options for -type are:
  - d A Directory
  - f A File
  - l A Link

Logical Operations

Searches using "find" may be created using multiple logical conditions connected using the logical operations ("AND", "OR" etc). By default, options are concatenated using AND. In order to have multiple search options connected using a logical "OR", the code is generally contained in brackets to ensure proper order of evaluation. For instance: `\(-perm -2000 -o -perm -4000 \)`. The symbol "!" is used to negate a condition (it means logical NOT). "NOT" should be specified with a backslash before exclamation mark (\!). For instance: `find \! -name "*.tgz" -exec gzip {} \;`. The `\( expression \)` format is used in cases where there is a complex condition. For instance: `find / -type f \( -perm -2000 -o -perm -4000 \) -exec /mnt/cdrom/bin/ls -al {} \;`.

Output Options

The find command can also perform a number of actions on the files or directories that are returned. Some possibilities are discussed below:

- "-print": The "print" option displays the names of the files on standard output. The output can also be piped to a script for post-processing. This is the default action.
- "-exec": The "exec" option executes the specified command. This option is most appropriate for executing moderately simple commands.

Find can execute one or more commands for each file it has returned using the "-exec" parameter. Unfortunately, one cannot simply enter the command. For instance:

- find . -type d -exec ls -lad {} \;
- find . -type f -exec chmod 750 {} ';'
- find . -name "*rc.conf" -exec chmod o+r '{}' \;
- find . -name core -ctime +7 -exec /bin/rm -f {} \;
- find /tmp -exec grep "search_string" '{}' /dev/null \; -print

An alternative to the `-exec` parameter is to pipe the output into the `xargs` command. This section has only just touched on find and it is recommended that the system security tester investigate this command further. A commonly overlooked aspect of the `find` command is in locating files that have been modified recently. The command:

```
find / -mtime -7 -print
```

displays files on the system recursively from the '/' directory up sorted by the last modified time. The command:

```
find / -atime -7 -print
```

does the same for last access time. When access is granted to a system and a file that is run, the file times change. Each change to a file updates the modified time and each time a file is executed or read, the last access time is updated. These (the last modified and accessed times) can be updated using the touch command.

A Summary of the find command

Effective use of the find command can make any security assessment much simpler. Some key points to consider when searching for files are detailed below:

- Consider where to search and what subtrees will be used in the command remembering that multiple piles may be selected
  - find /tmp /usr /bin /sbin /opt -name sar
- The find command allows for the ability to match a variety of criteria
- -name: search using the name of the file(s). This can be a simple regex.
- -type: what type of file to search for (d -- directories, f -- files, l -- links)
- -fstype typ: allows for the capability to search a specific filesystem type
- -mtime x: File was modified "x" days ago
- -atime x: File was accessed "x" days ago
- -ctime x: File was created "x" days ago
- -size x: File is "x" 512-byte blocks big
- -user user: The file's owner is "user"
- -group group: The file's group owner is "group"
- -perm p: The file's access mode is "p" (as either an integer/symbolic expression)

Think about what you will actually use the command for and consider the options available to either display the output or send it to other commands for further processing:

- -print: display pathname (default)
- -exec: allows for the capability to process listed files ({} expands to current found file)
- Combine matching criteria (predicated) into complex expressions using logical operations -o and -a (default binding) of predicates specified.

CRAIG S. WRIGHT
CraigSWright@acm.org

Great Specials

On FreeBSD® & PC-BSD® Merchandise

Give us a call & ask about our
SOFTWARE BUNDLES
1.925.240.6652

$39.95
FreeBSD 9.1 Jewel Case CD Set
or FreeBSD 9.1 DVD

$29.95
PC-BSD 9.1 DVD

$49.95 Save a BUNDLE!
The PC-BSD 9.0 Users Handbook
PC-BSD 9.1 DVD

$99.95
The FreeBSD CD or DVD Bundle
Inside each CD/DVD Bundle, you'll find:
FreeBSD Handbook, 3rd Edition
Users Guide FreeBSD Handbook, 3rd Edition, Admin Guide
FreeBSD 9.1 CD or DVD set
FreeBSD Toolkit DVD

Stylish Dress Attire
Look Your Professional Best

Comfy Apparel
Stay Warm in Zip Ups & Pullovers

T-Shirts
Lots of Styles to Choose From

FreeBSD 9.1 Jewel Case CD/DVD $39.95

CD Set Contains:

- **Disc 1** Installation Boot LiveCD (i386)
- **Disc 2** Essential Packages Xorg (i386)
- **Disc 3** Essential Packages, GNOME2 (i386)
- **Disc 4** Essential Packages (i386)

FreeBSD 9.0 CD $39.95
FreeBSD 9.0 DVD $39.95

FreeBSD Subscriptions

Save time and $$$ by subscribing to regular updates of FreeBSD

FreeBSD Subscription, start with CD 9.1 .... $29.95
FreeBSD Subscription, start with DVD 9.1 .... $29.95
FreeBSD Subscription, start with CD 9.0 .... $29.95
FreeBSD Subscription, start with DVD 9.0 .... $29.95

PC-BSD 9.1 DVD (Isotope Edition)

PC-BSD 9.1 DVD $29.95
PC-BSD Subscription $19.95

The FreeBSD Handbook

The FreeBSD Handbook, Volume 1 (User Guide) $39.95
The FreeBSD Handbook, Volume 2 (Admin Guide) $39.95

The FreeBSD Handbook Specials

The FreeBSD Handbook, Volume 2 (Both Volumes) $59.95
The FreeBSD Handbook, Both Volumes & FreeBSD 9.1 .... $79.95

PC-BSD 9.0 Users Handbook $24.95

BSD Magazine $11.99

The FreeBSD Toolkit DVD $39.95

FreeBSD Mousepad $10.00

FreeBSD & PCBSD Caps $20.00

BSD Daemon Horns $2.00

Bundle Specials!
Save $$$

Just Plain Fun
Mousepads & Novelty Horns

BSD Magazine
Available Monthly

FreeBSD Mall

For even MORE items
visit our website today!
www.FreeBSDMall.com

Getting to Grips With the Gimp – Part 7

In the seventh part in our series on the Gimp we will retouch an old photo and start working with filters.

What you will learn…

- How to manipulate images like a design pro

What you should know…

- General PC administration skills

Retouching photos is a common request, and we will recreate the Stalinist propaganda of the removal of Nikolai Yezhov from the photo with Stalin to illustrate how to manipulate old images. We will then apply some filters to enhance the photograph.

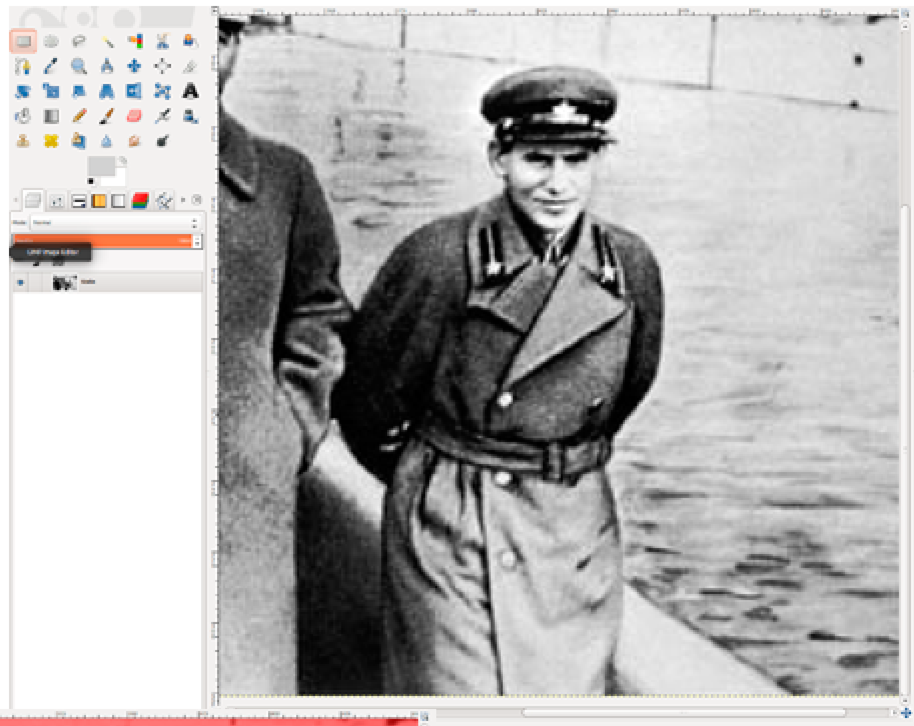
Download the image listed in Table 1.

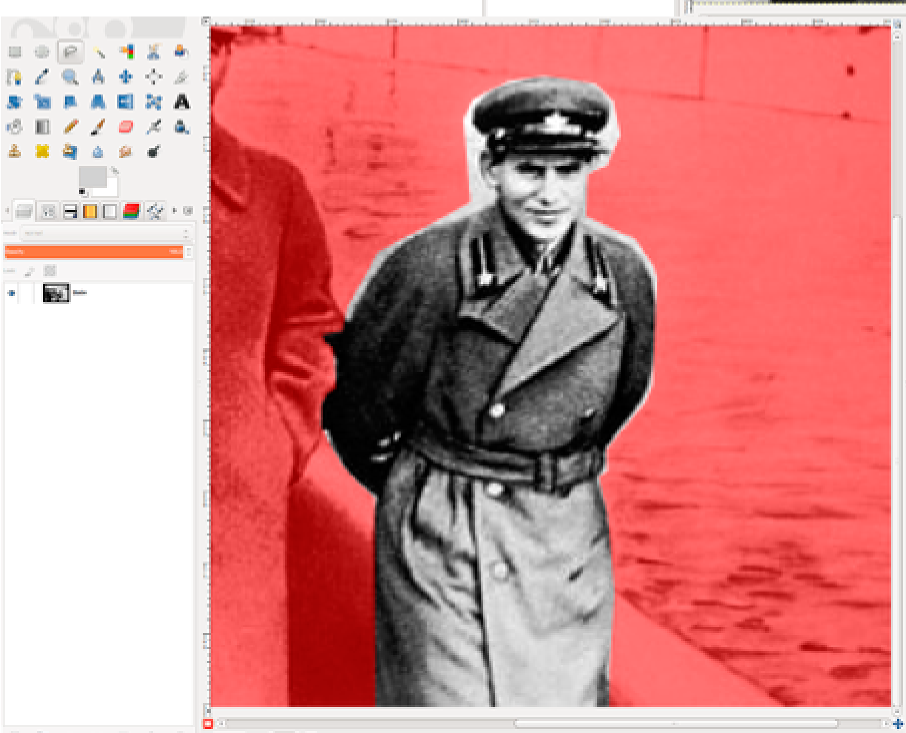
Table 1. *Details and credits*

Image	URL	Details and credits
The Commissar Vanishes	http://upload.wikimedia.org/wikipedia/commons/9/91/Voroshilov%2C_Molotov%2C_Stalin%2C_with_Nikolai_Yezhov.jpg	Wikimedia – Voroshilov, Molotov, Stalin, with Nikolai Yezhov

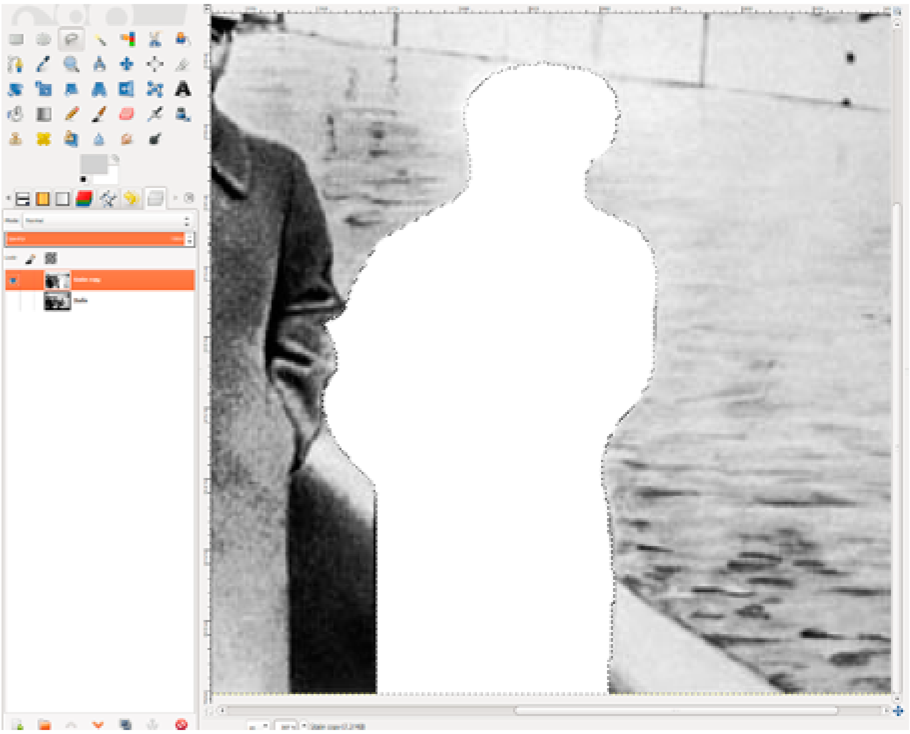
Duplicate the original layer, hide the original and select the second layer.

Zoom in on the figure of Yezhov and select him using the free select tool. I have turned the quick mask on to highlight the area selected [Figure 1 and Figure 2].

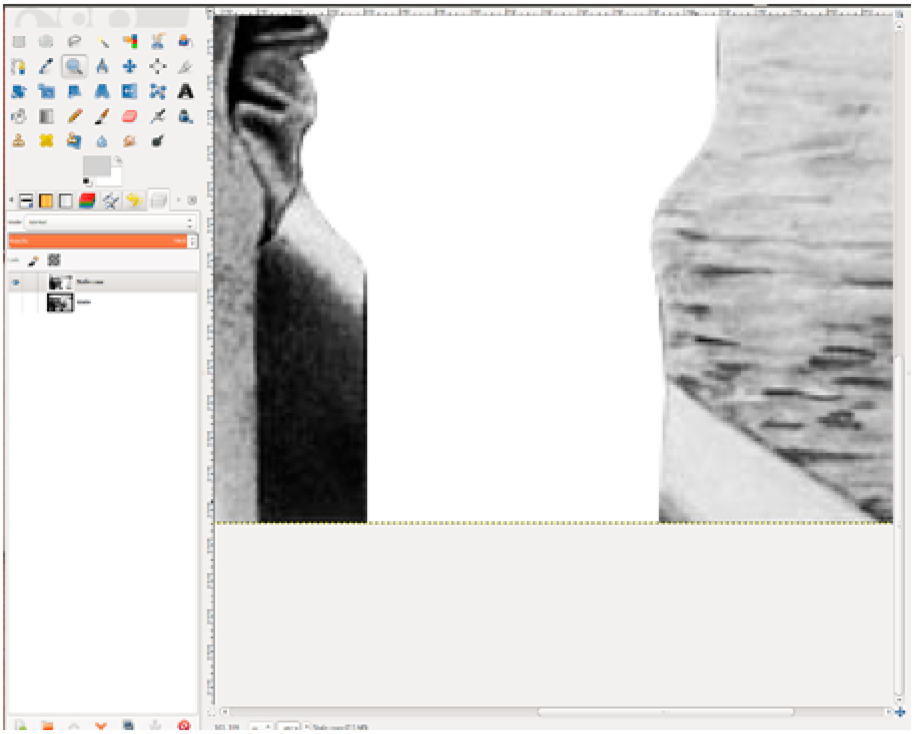




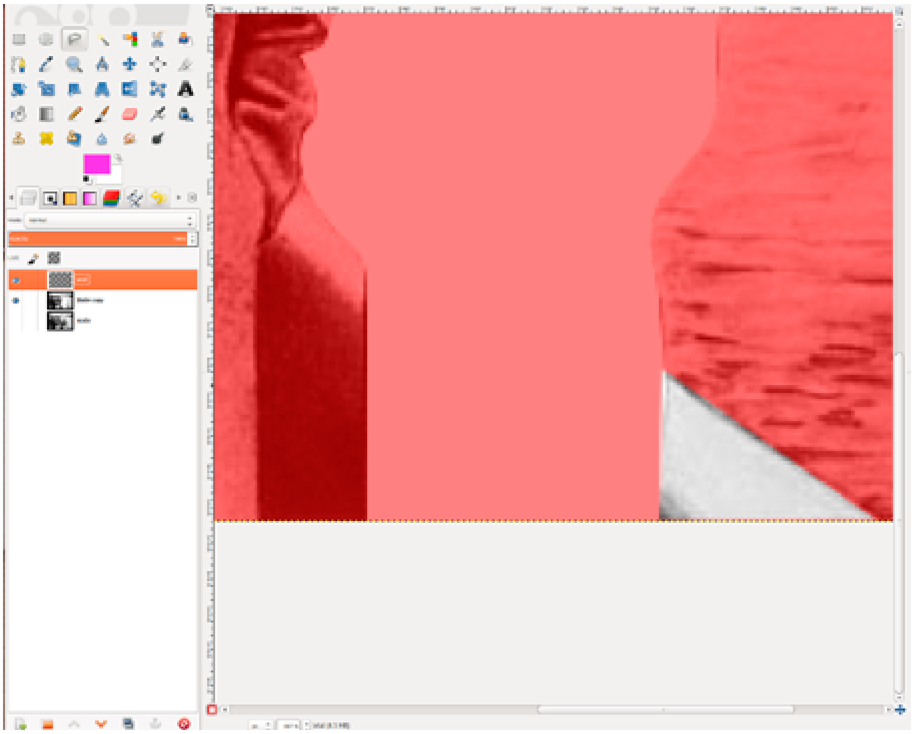
Press DEL to remove Yezhov [Figure 3].



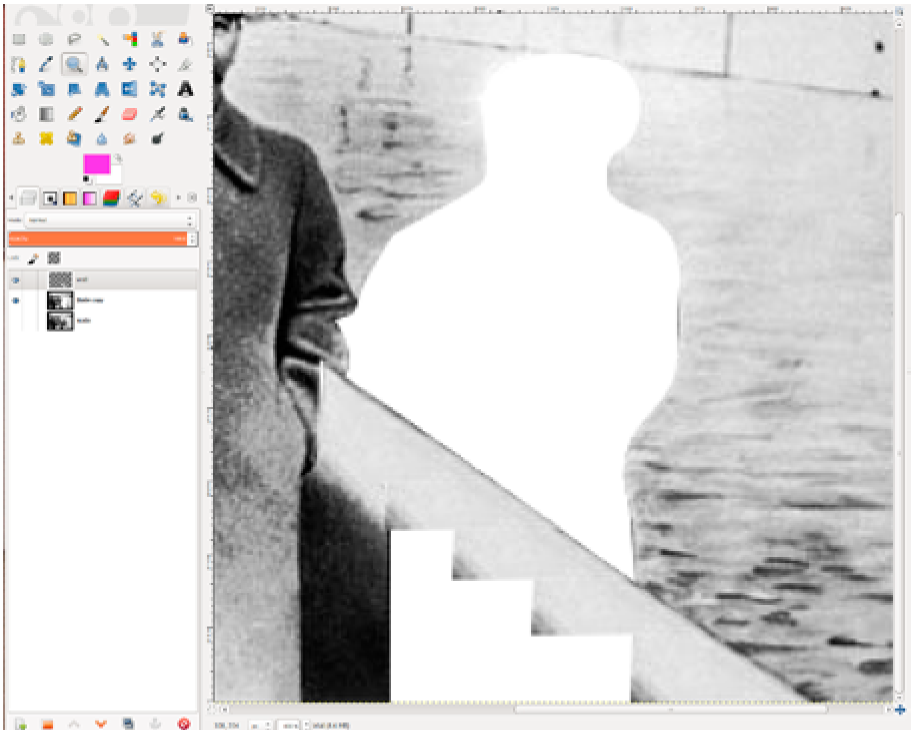
Press *Ctrl Shift A* to deselect the area and zoom in on the bottom right hand side of the wall [Figure 4].



Select and copy the area of wall at the bottom right of the photo as a patch. Create a new transparent layer called *wall* and select it [Figure 5].

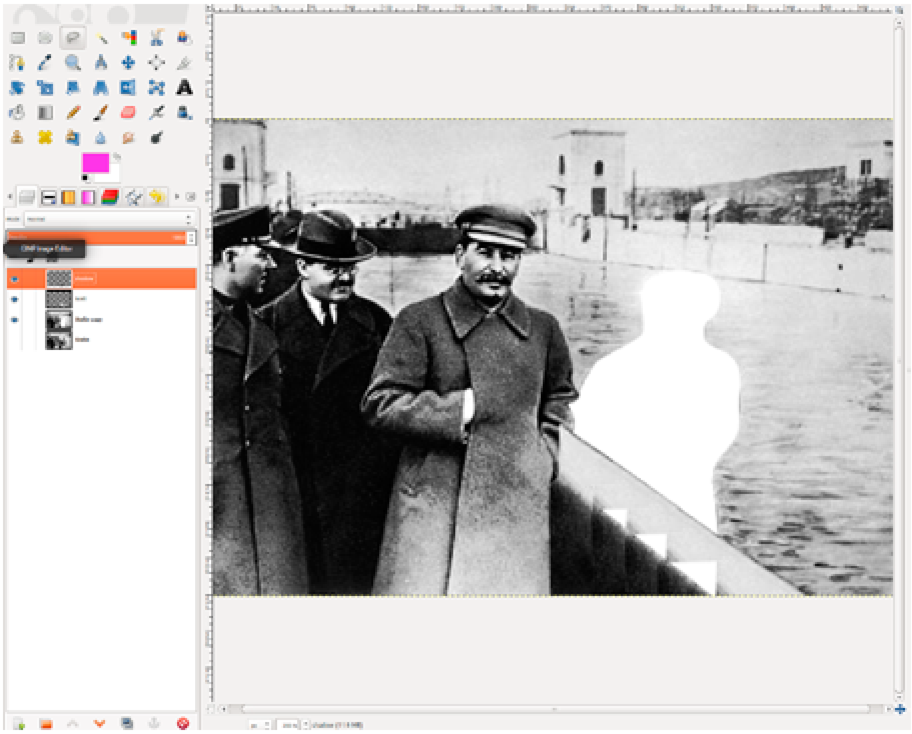


Following the line of the wall into the horizon, repeatedly paste the section of wall up past Stalin's arm [Figure 6].



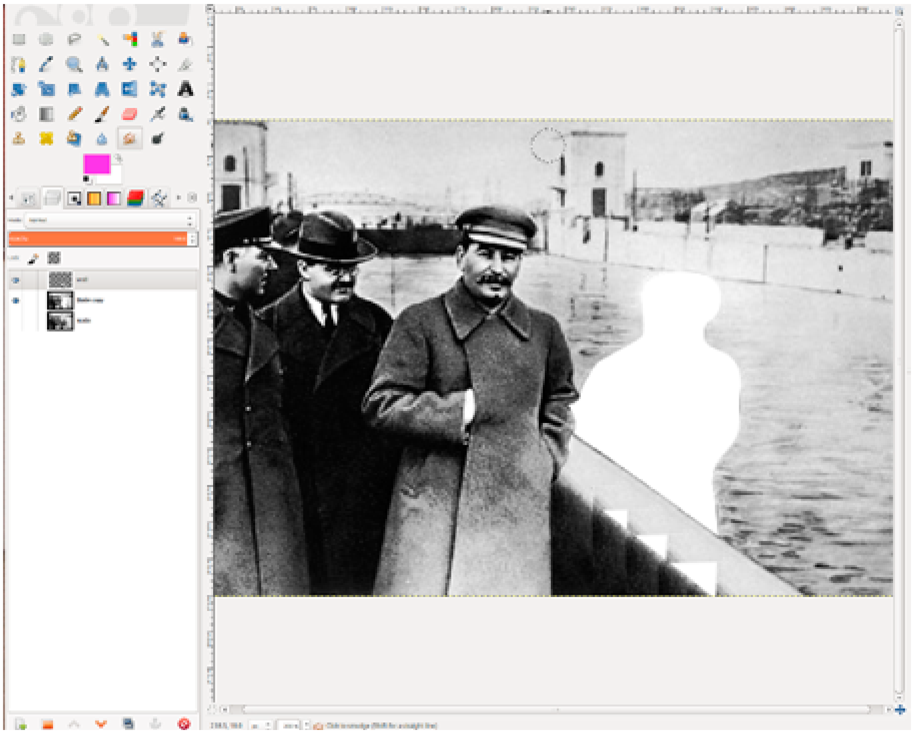
Temporarily hide the wall layer and select a suitable area of the wall shadow from the lower layer. Create a new transparent layer called *shadow* above *wall.*

Click on shadow and repeat the process in step 7 to duplicate the selected patch from Stalin's elbow to the lower corner of the picture [Figure 7].

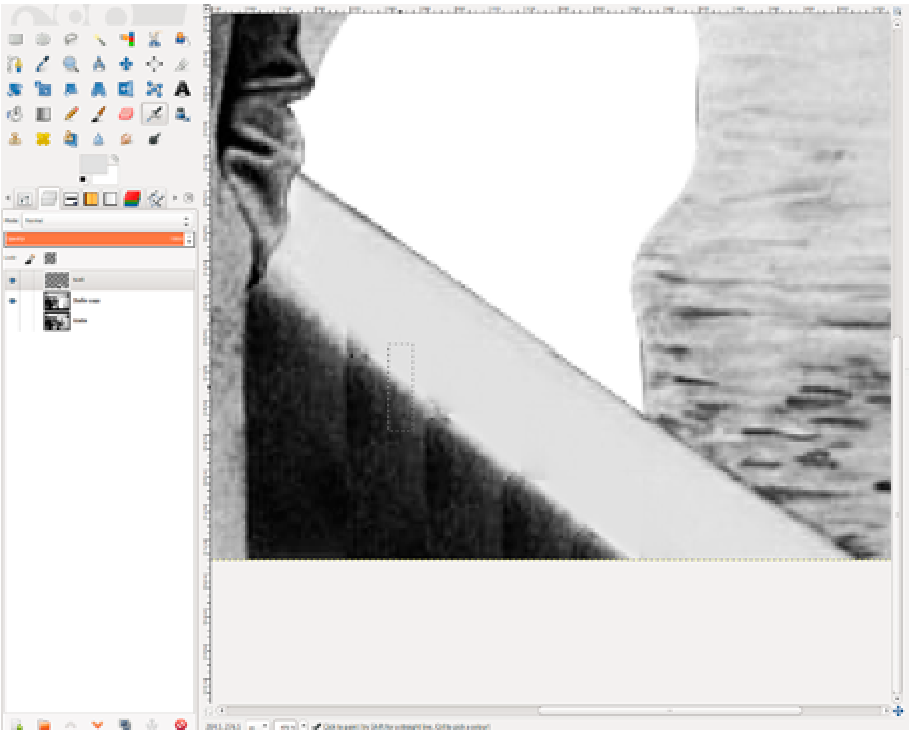


Merge the *Shadow* and *Wall* layers by right clicking on *Shadow* and choose *Merge Down.*

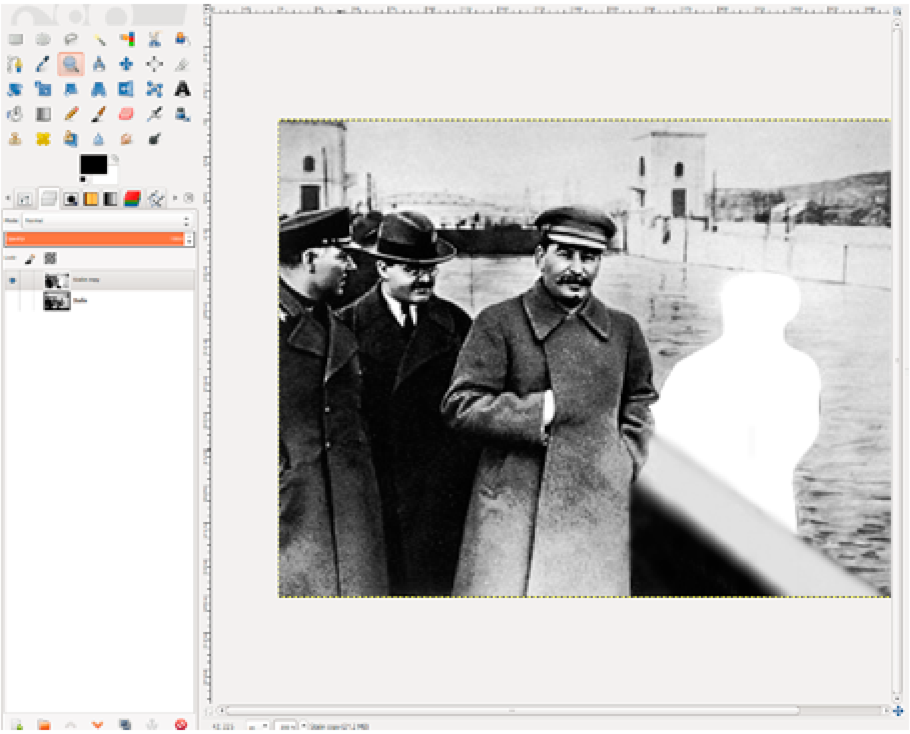
Give this layer 50% opacity, delete the overlap on Stalin's left arm and return the opacity to 100% [Figure 8].



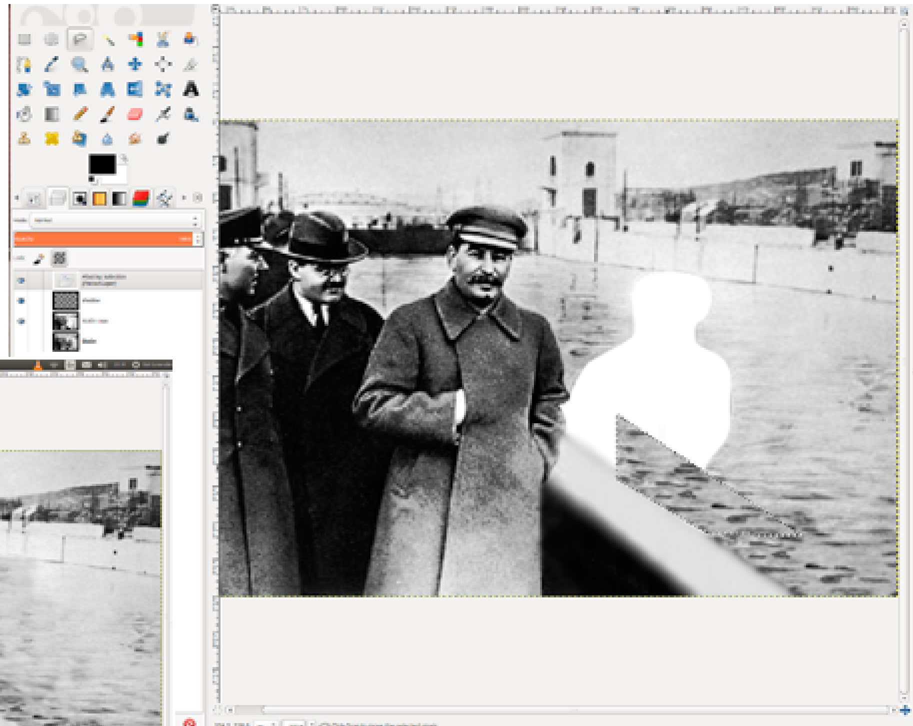
Use the block brush (Colour E0E0E0) rotated 90 degrees to airbrush any gaps on the top part of the wall [Figure 9].

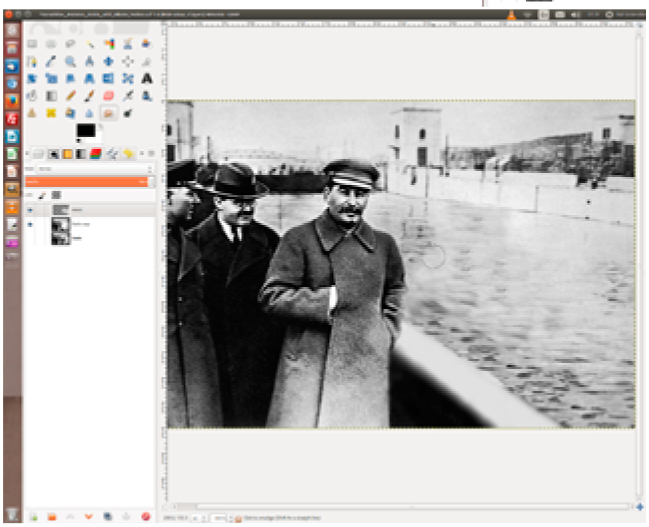


Create a new layer and using the airbrush tool touch up the shadow. Adjust the opacity of the layer to get a realistic appearance and merge the layer twice. Run the blur tool lightly over the sharp edges of the wall. Save the image [Figure 10].

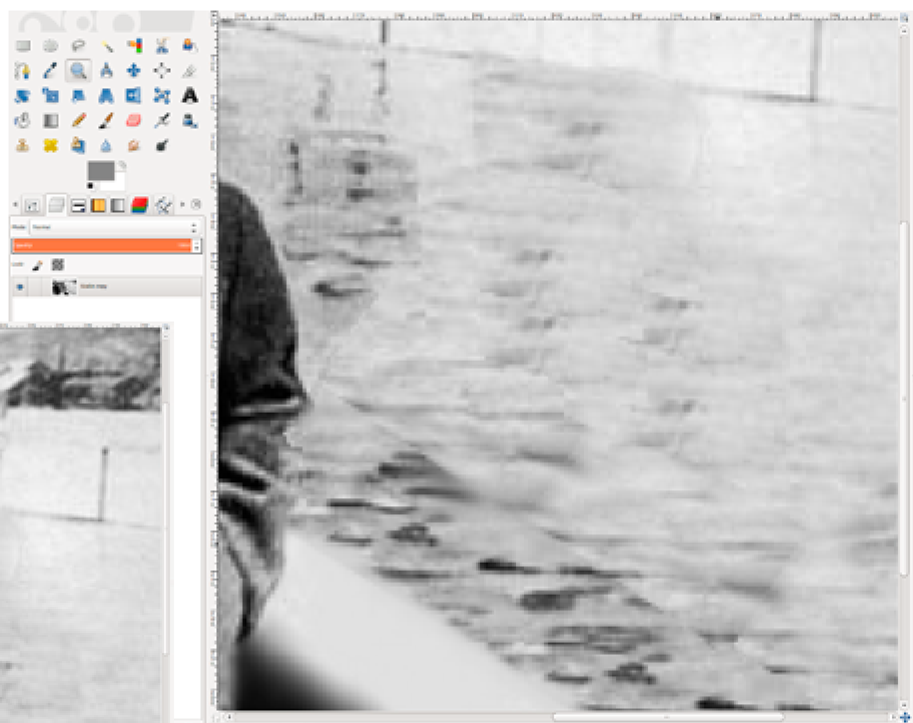


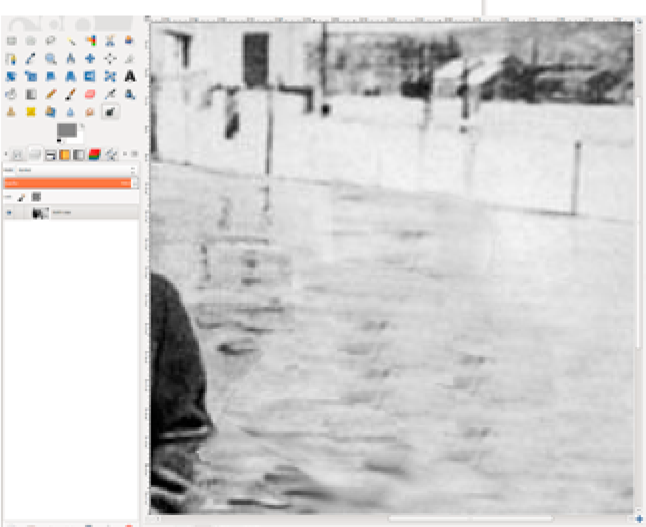
Create a new transparent layer called water and select a foreground patch of water from the lower layer. Copy and paste this into the new layer and work towards Stalin's arm. Repeat slightly higher up from the bottom right of the image to get a realistic water pattern. Use the [Figure 11 and 12]



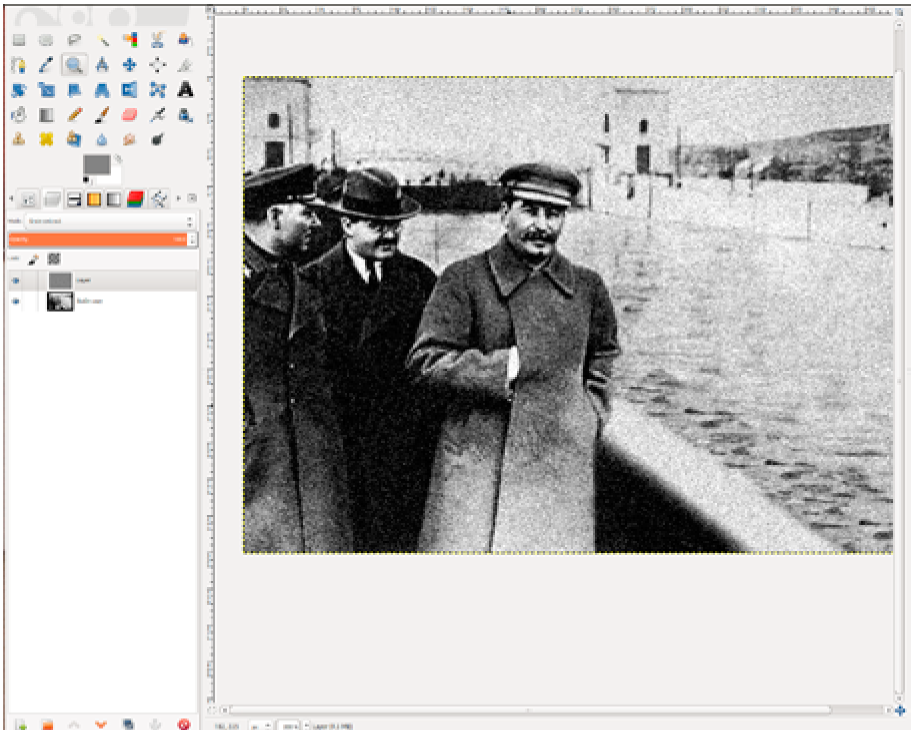


Under close examination, the modifications would not pass close examination. In particular although the wave pattern is fine, the contrast varies and is patchy. Touch this up with the dodge tool, adjusting the brush shape and size as required [Figure 13 & 14].



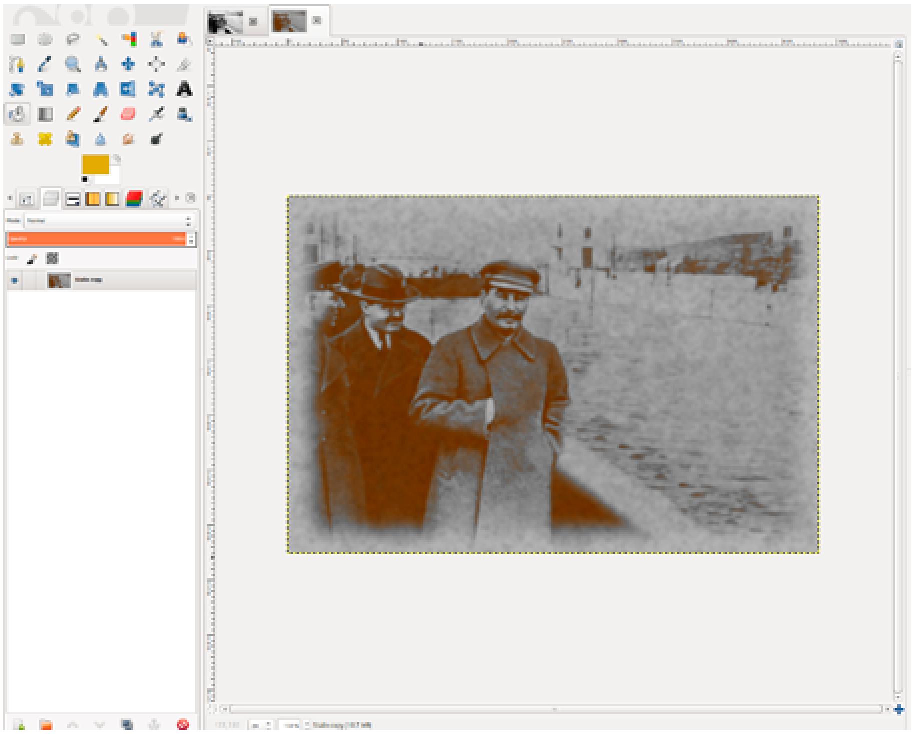


Add a new layer and fill this with 50% grey (828282). From the filters dialogue, select Noise → RGB noise. From the Colours menu, desaturate by luminosity. Then change the new layer mode to grain extract. This will result in a grainy image [Figure 15].



To create a sepia toned image, disable the grain layer and select Filters → Decor → Old photo [Figure 16].

Congratulations, you can now join the ranks of the propagandists.



ROB SOMERVILLE

Rob Somerville has been passionate about technology since his early teens. A keen advocate of open systems since the mid-eighties, he has worked in many corporate sectors including finance, automotive, airlines, government and media in a variety of roles from technical support, system administrator, developer, systems integrator and IT manager. He has moved on from CP/M and nixie tubes but keeps a soldering iron handy just in case.

Over twenty years ago, Mitchell Kapor, the founder and former C.E.O. of Lotus Development Corporation stated, "The question of what to do about Microsoft is going to be a central public policy issue for the next 20 years. Policy makers don't understand the real character of Microsoft yet". Has government and industry smelled the coffee yet?

I am writing this column on the last day of my annual holiday and to be perfectly honest, I am not looking forward to going back in the trenches again. Just before going on leave, our organisation had an outbreak of the Cryptowall virus, and while we have managed to contain this professionally, in one day alone our external firewall rejected over 5,000 infected attachments. To add insult to injury, my broadband line at home has being playing up since late last year culminating in spasms of slow and intermittent connectivity and no land-line, sometimes extending to periods of over 10 days. In a final sop to the front line technical support desk, I have relocated all my IT kit downstairs next to the master telephone socket to "prove" the fact that it is not our internal house wiring – much to the chagrin of my long suffering wife. The final straw came last Friday, when after 10 hours of downtime I phoned my ISP for the umpteenth time just for the ADSL to come back up again while being held in the telephone queue. Now to give my ISP their due, they have always been extremely apologetic, helpful and efficient – but the core problem has lain with the infrastructure provider. I have been promised an engineer visit on Friday so hopefully we will get to the root of the problem, but I have my reservations. Unless the line decides to exhibit a fault during the tests, the usual cat and mouse game of trying to trace and isolate an intermittent fault will continue *ad nauseum*.

And so be it with the scammers who are currently behind the current wave of encryption malware. Having examined the offending exe's with a hex editor, part of the payload embeds the path of the infected user directory in the exe itself thereby circumventing any detection via checksum, which is a common method used by anti-virus software. So from the hacker perspective for every machine infected, you potentially have a zero-day exploit at hand to mail out to another tranche of victims. With mass-mailer malware regularly included in the latest generation of viruses, this is a pernicious form of attack. Unless your anti-virus software is set to paranoid mode, there is a good chance something nasty will get past. And of course, this does not prevent users from accidentally or foolishly clicking and opening an infected attachment disregarding any warning messages. Or in the case of a particularly demented user I once encountered, turning off anti-virus software altogether as it slowed down his PC.

As I have stated on many occasions, trying to get Law Enforcement (in the UK at least) to take an interest in this is virtually impossible, unless of course you have been defrauded in which case you are urged to contact them. With small businesses becoming victims it is not unheard of for people to pay the scammers to get the encryption codes back, in which case if the decryption process is successful it results in a very interesting paradox – the scammers' reputation rises if they honour their word. In theory, they

have fulfilled contract law, which is an astute move as the penalties under civil law can be greater than under criminal law and the burden of proof is based on the balance of probabilities rather than beyond all reasonable doubt. So proving fraud if you make payment potentially could be difficult. The fact that this whole distasteful scenario takes place across international boundaries with different legal systems over an encrypted network and you can understand why the police are so slow to act. So where should we point the finger?

If we were to design doors and windows for housing that let rain and wind in, nobody would buy them. Unless of course I was a virtual monopoly, and then there would be great outcry to the policy makers about "Fitness for purpose". Another better supplier would be appointed and I would in reality face bankruptcy. Unfortunately, Microsoft has had all the benefits of a trojan horse, and decades have passed so it is now in the magical alternative universe of being "Too big to fail". Vested interest has firmly cemented Microsoft's feet under the corporate and domestic table. Likewise with the telcos – as an individual I have a snowball in hell's chance of suing for breach of contract due to lack of service provision; all the small print in the Service Level Agreements has seen to that. So as the "little man" in reality I don't have much choice as far as infrastructure is concerned apart from maybe moving houses.

50 years ago, the British Prime Minister Harold Wilson called for Britain to be forged in the "white heat" of a technological and scientific revolution. We have got that all right – but it is the anarchists rather than the citizens that are driving change. Software security vendors are calling for a form of Internet Identification – but as any intelligent person knows this will play into the hands of the fraudsters – rather than stealing passports and drivers' licences, Internet User ID's will be the theft of choice for the criminal, all the while their co-conspirator Microsoft will plead innocence. If you think I am being harsh, in law Microsoft's role could be considered "Aiding and abetting" under English law, and "Accessory before the fact" under an American jurisdiction.

We have reached the point of critical mass, the tipping point where the perfect storm of greed and convenience has won over strong engineering design principles. Microsoft is not the only culprit here; Adobe (and no doubt many others) have embedded themselves so deeply inside the operating system that they are virtually inseparable. Having understood the issue of regular cookies, a well educated computer-literate friend of mine was shocked when I explained to him the scope and use of Flash cookies. And people wonder why the concepts of encryption, privacy and security are gaining traction and publicity of late. For too long the corporates have ignored these principles and now the chickens are coming home to roost. The Internet and use of Internet connected devices is now so ingrained and essential to smooth running of a modern society that it is on par with the mission-critical status of the electrical, water, fuel, and food chains. If any of these services break down, optimistically you have a window of 7 days maximum before civil unrest takes over. Here in the UK, benefit reforms are forcing claimants to register and apply for jobs online. And yet we do not have a secure and reliable technology platform other than those based on Open Source, and it could be argued (rightly so) that the only reason that it does not fall foul of the criminals is due to lack of consumer footprint. I hope and pray that it does not happen, but at the current level of attack it will not be long before something major collapses – whether this be the widespread compromise of a bank, government agency, media outlet or health provider *et al*. It is already happening piecemeal on an individual basis, as happened recently with the failure of the Adobe Creative Cloud (which was not security related) unlike the theft of the encrypted credit card details for their 38 million subscribers. All it will take is sloppy management and a particularly vicious payload. It matters not the reason – financial exploitation, sheer will-full destructiveness or technological warfare on the part of a hostile country – the damage will have been done.

The biggest virus of all, a colleague wryly commented *is* the Microsoft operating system. Once the penny drops in the marketplace that current software and infrastructure solutions are no longer sufficient to keep us or our data safe and that we have a global single point of failure – technology – the future for MSC and other culprits will be bleak. The Emperors' clothes will be seen for what they are, and both the anger and the response of the consumer will be unprecedented. We have until then to address this by legislation, law enforcement, innovation and design unless we want Information Technology as an industry to be relegated to the ethical trash can like politicians, estate agents, lawyers and used car salesmen.

ROB SOMERVILLE

Rob Somerville has been passionate about technology since his early teens. A keen advocate of open systems since the mid-eighties, he has worked in many corporate sectors including finance, automotive, airlines, government and media in a variety of roles from technical support, system administrator, developer, systems integrator and IT manager. He has moved on from CP/M and nixie tubes but keeps a soldering iron handy just in case.